

Gemini Chat Export

Exported on: 16/10/2025 19:17:06

You

rédige nous une documentation de défense pour notre infrastructure, nous avons une infra contenrisé et redéployable avec terraform et ansible, aide nous a ajouter différent moyen de prévention, detection d'intrusion et protection générale en donnant des cas pratique concret pour chaque proposition. La défense aux intrusions sera également récompensée, TOUS les coups sont permis pour la défense ! Nous avons déjà réfléchi aux différentes défenses possibles et les défenses les plus ingénieuses marqueront plus de points ! Une documentation de chaque action vous sera demandée et devra être apporter au conseil des ministres afin qu'il vous attribue le nombre de points mérités. nous sommes dans le cas d'une école type poudlard

Gemini

Absolument. Voici une documentation de défense pour votre infrastructure, conçue pour être présentée à un "Conseil des Ministres". L'approche est holistique, combinant prévention, détection et des contre-mesures offensives, le tout dans l'univers thématique de Poudlard.

DOCUMENTATION DE DÉFENSE STRATÉGIQUE DE L'INFRASTRUCTURE NUMÉRIQUE DE POUDLARD

À l'attention du Conseil des Ministres de la Magie Numérique,

Le monde des Moldus évolue, et avec lui les menaces qui pèsent sur notre illustre école. Nos ennemis, les Mages Noirs du web, cherchent constamment à percer nos défenses pour dérober nos secrets (données), perturber nos enseignements (services) ou ternir notre réputation.

Notre infrastructure actuelle, bâtie sur la modularité des **Salles sur Demande** (conteneurs) et déployable à volonté grâce à notre **Charte de Construction Magique** (Terraform) et notre **Livre des Sortilèges d'Automatisation** (Ansible), est une merveille de technologie magique. Cependant, une forteresse, aussi bien conçue soit-elle, doit être protégée par de puissants enchantements.

Nous vous présentons ici notre stratégie de défense en trois actes : les **Sortilèges de Prévention**, l'**Œil Magique de Maugrey** pour la détection, et enfin, notre programme le plus audacieux : le **Département des Mystères**, où tous les coups sont permis.

I. Les Sortilèges de Prévention (Prevention)

La meilleure des batailles est celle que l'on n'a pas à livrer. Ces enchantements de base visent à rendre notre château numérique aussi impénétrable que possible.

Sortilège n°1 : Le Charme du Repousse-Moldu (Segmentation Réseau)

Le principe est simple : un élève de Serpentard ne devrait pas pouvoir entrer dans la salle commune de Gryffondor. De la même manière, nos services ne devraient communiquer qu'avec ceux qui sont strictement autorisés.

- **Cas Pratique Concret :** Nous utilisons **Calico** pour tisser des politiques réseau (Network Policies) au sein de notre cluster Kubernetes. Par défaut, aucun conteneur ("Salle") ne peut en contacter un autre. Nous créons ensuite des règles explicites.
 - **Exemple :** Le conteneur **Portail-Gryffondor** (frontend web) peut communiquer avec l'API **Bibliothèque-Serdaigle** sur le port 443, mais est incapable de joindre la base de données **Chaudron-Poufsouffle**.
 - **Implémentation :** Un playbook **Ansible** déploie la **NetworkPolicy** suivante après que **Terraform** a créé le cluster.

```
yaml      apiVersion:
networking.k8s.io/v1      kind: NetworkPolicy      metadata:
name: allow-frontend-to-api      namespace: bibliotheque
spec:      podSelector:      matchLabels:      app:
api-serdaigle      ingress:      - from:      -
podSelector:      matchLabels:      app:
portail-gryffondor      ports:      - protocol:
TCP      port: 443
```

Sortilège n°2 : Le Coffre-fort de Gringotts (Gestion des Secrets)

Un mot de passe écrit sur un parchemin est un mot de passe volé. Nos secrets (clés d'API, mots de passe de base de données, certificats) doivent être gardés dans un coffre-fort inviolable.

- **Cas Pratique Concret :** Nous déployons **HashiCorp Vault**. Les applications n'ont plus de secrets en clair dans leurs fichiers de configuration. Au démarrage, elles s'authentifient auprès de Vault avec une identité éphémère et récupèrent des secrets dynamiques qui expirent après quelques heures.
 - **Exemple :** Le conteneur de notre application, au lieu d'avoir un mot de passe de base de données, possède un rôle Vault. Il demande à Vault : "Je suis l'application **Hibou-Express**, donne-moi un accès

- à la base de données.” Vault génère un nom d'utilisateur et un mot de passe uniques avec une durée de vie de 1 heure et les lui transmet.
- **Implémentation : Terraform** provisionne le serveur Vault et configure les politiques d'accès. **Ansible** configure les applications pour qu'elles pointent vers l'adresse de Vault.

Sortilège n°3 : Le Sceau d'Immutabilité (Images de Conteneurs Durcies)

Une potion une fois brassée et scellée ne doit plus jamais être modifiée. Il en va de même pour nos images de conteneurs. Nous créons des “Golden Images” qui sont scannées, signées et rendues non modifiables.

- **Cas Pratique Concret** : Notre pipeline CI/CD (avec GitLab CI, par exemple) suit ces étapes :
 1. Construction de l'image du conteneur.
 2. Scan de l'image avec **Trivy** pour y déceler toute trace de magie noire (vulnérabilités connues).
 3. Si le scan est réussi, l'image est signée numériquement avec **Cosign/Sigstore**, prouvant son origine et son intégrité.
 4. L'image est poussée vers notre registre privé sécurisé, **Harbor**.
- **Implémentation : Ansible** et notre orchestrateur de conteneurs sont configurés pour refuser de déployer toute image qui ne porte pas une signature cryptographique valide. C'est le sortilège *Cave Inimicum* de notre chaîne d'approvisionnement logicielle.

II. L'Œil Magique de Maugrey (Detection)

Même les meilleurs murs peuvent avoir des fissures. L'Œil Magique voit tout : les menaces visibles et invisibles, et nous alerte avant qu'il ne soit trop tard.

Outil n°1 : Le Détecteur de Présences Indésirables (Détection d'Intrusion en Temps Réel)

Cet œil magique observe le comportement à l'intérieur de nos conteneurs et nous alerte dès qu'une activité anormale se produit.

- **Cas Pratique Concret** : Nous déployons **Falco** comme un gardien dans chaque recoin de notre château (en DaemonSet sur les nœuds Kubernetes). Falco surveille les appels système et les compare à un jeu de règles.
 - **Exemple de règle** : “Alerte de haute priorité ! Un Mangemort (un shell) vient d'apparaître dans le conteneur **Salle-des-Trophées** (un conteneur de production qui n'a aucune raison d'avoir un shell interactif).” ou “Quelqu'un tente de modifier le Grimoire des Comptes (/etc/passwd) !”

- **Implémentation** : Une recette **Ansible** déploie Falco et nos règles personnalisées, qui envoient les alertes critiques directement dans la salle de crise des Aurores (un canal Slack ou PagerDuty).

Outil n°2 : La Pensine (Centralisation et Analyse des Logs)

Chaque événement, chaque porte qui grince, chaque sortilège lancé dans notre infrastructure laisse une trace (un log). La Pensine nous permet de collecter tous ces souvenirs et de les analyser pour y trouver des schémas d’attaque.

- **Cas Pratique Concret** : Nous mettons en place une pile **EFK** (**Elasticsearch**, **Fluentd**, **Kibana**).
 - **Fluentd** agit comme un elfe de maison, collectant les logs de tous les conteneurs et de tous les services.
 - **Elasticsearch** stocke et indexe ces millions de souvenirs.
 - **Kibana** nous permet de plonger dans la Pensine et de visualiser les événements. Nous créons des tableaux de bord pour surveiller en temps réel les tentatives de connexion échouées, les erreurs applicatives suspectes, et les alertes de nos autres systèmes de défense.
 - **Implémentation** : **Terraform** peut provisionner les ressources nécessaires (VMs, stockage) et **Ansible** déploie et configure la pile EFK.

III. Le Département des Mystères (Active Defense & Contre-Attaque)

Pour les menaces les plus avancées, une défense passive ne suffit plus. Le Département des Mystères est notre laboratoire de contre-espionnage et de défense non conventionnelle. **Ici, nous prenons l’initiative.**

Initiative n°1 : La Chambre des Secrets (Honeypot)

Pourquoi se contenter de bloquer un attaquant quand on peut l’attirer dans un piège pour l’étudier ? Nous créons une fausse partie de notre infrastructure, irrésistiblement attrayante, mais entièrement sous notre contrôle.

- **Cas Pratique Concret** : Nous déployons un conteneur se faisant passer pour une vieille base de données non patchée, contenant de fausses listes d’élèves et des recettes de potions sans valeur. Ce “honeypot” est isolé du reste du réseau. Chaque interaction avec lui est enregistrée : les commandes tapées, les fichiers uploadés, les techniques utilisées.
 - **Exemple** : Un attaquant pense avoir trouvé une faille. Il se connecte, liste les bases de données, tente d’exfiltrer le fichier `Recette_Felix_Felicitis.sql`. À cet instant, nous recevons une alerte majeure avec son adresse IP, et nous pouvons observer en

direct ses moindres faits et gestes, apprenant ainsi ses méthodes pour renforcer nos vraies défenses.

- **Implémentation** : Des outils comme **Kube-honeypot** ou des services dédiés sont déployés via **Ansible**. Les logs sont envoyés à une section spéciale de notre Pensine pour analyse forensique.

Initiative n°2 : Le Sortilège du Portoloin Piégé (Canary Tokens)

Nous laissons traîner des “objets” magiques dans notre infrastructure. Si un voleur les emporte, ils nous signalent immédiatement sa position.

- **Cas Pratique Concret** : Nous générons des **Canary Tokens** et les disséminons :
 1. Une fausse clé d'API AWS dans un fichier de configuration : **AKIA-FAKE-KEY-FOR-ATTACKER**. Si l'attaquant tente de l'utiliser, nous sommes alertés.
 2. Un document Word nommé **Plan_defense_Poudlard_Confidentiel.docx** sur un partage réseau. Le document est vide, mais son ouverture déclenche une alerte.
 3. Une URL unique dans un fichier de code source en commentaire : `# TODO: remove this temporary admin panel link: http://poudlard.dev/temp-admin-XYZ`. Cliquer sur ce lien nous informe qu'un curieux a fouillé notre code.
- **Implémentation** : Ces tokens sont injectés dans nos configurations et nos conteneurs via les playbooks **Ansible** lors du déploiement. C'est une défense simple, mais diaboliquement efficace.

Initiative n°3 : Le Sortilège de Stupéfixion (Blocage Réactif)

Quand un ennemi attaque de manière répétée et prévisible, nous ne nous contentons pas de parer ses coups. Nous le neutralisons.

- **Cas Pratique Concret** : Nous utilisons **CrowdSec**, un système de sécurité collaboratif et automatisé. Il analyse les logs de nos services (WAF, serveurs web, SSH).
 - **Exemple** : Une adresse IP tente une attaque par force brute sur notre page de connexion. Après 5 tentatives échouées, CrowdSec détecte ce comportement agressif. Il communique cette information à notre pare-feu, qui applique un **DROP** (un “Stupefy!”) sur cette IP pour les 24 prochaines heures. L'attaquant est neutralisé avant d'avoir pu deviner le premier mot de passe.
 - **Implémentation** : L'agent CrowdSec est déployé via **Ansible** sur nos serveurs. Le “bouncer” (l'exécuteur du blocage) est également installé sur notre pare-feu ou notre reverse proxy.

Initiative n°4 : Le Charme de Confusion (Tarpitting & Deception)

Notre arme la plus ingénieuse. Nous ne nous contentons pas de bloquer l'ennemi ; nous le rendons fou. Nous jouons avec ses nerfs, nous lui faisons perdre son temps et nous le forçons à révéler ses outils.

- **Cas Pratique Concret :** Quand un attaquant scanne notre réseau, au lieu de voir des ports fermés, il voit des ports ouverts qui répondent avec une lenteur extrême (technique du “**tarpit**” ou pot de goudron). Une simple requête qui devrait prendre une milliseconde en prend 30. Un scan complet de notre infrastructure lui prendra des jours, voire des semaines, au lieu de quelques minutes. Pendant ce temps, chaque connexion lente est une alerte qui nous permet de l'étudier.
 - **Exemple avancé :** Pour un attaquant tentant d'exploiter un service qui n'existe pas, nous pouvons lui renvoyer des bannières de service erronées et des messages d'erreur conçus pour le tromper, le menant sur de fausses pistes et le poussant à essayer des exploits qui révéleront ses véritables intentions.
 - **Implémentation :** Des configurations spécifiques au niveau du pare-feu et des outils comme **LaBrea** peuvent être mis en place avec des scripts **Ansible**. C'est le summum de la défense active et psychologique.

Conclusion

Honorables membres du Conseil, cette stratégie transforme notre infrastructure d'une simple forteresse en un **château vivant et conscient**. Par la prévention, nous érigeons des murs enchantés. Par la détection, nous nous dotons d'une vigilance infaillible. Et par la défense active, nous transformons notre château en un labyrinthe hostile et mortel pour quiconque oserait s'y aventurer avec de mauvaises intentions.

L'automatisation via **Terraform** et **Ansible** garantit que ces défenses sont intégrées, reproductibles et constamment améliorées. Nous ne construisons pas seulement un mur ; nous entraînons une armée de sortilèges prête à défendre Poudlard contre les forces obscures du monde numérique.

Nous sollicitons votre approbation pour mettre en œuvre ces enchantements et garantir la pérennité de notre école pour les siècles à venir.

You

donne une formulation professionnelle et exporte le texte sur docs

Gemini

Absolument. Voici une version professionnelle de la documentation de défense, prête à être présentée.

Je ne peux pas exporter directement le texte vers Google Docs, mais vous pouvez simplement copier-coller le contenu ci-dessous dans un nouveau document.

DOCUMENTATION STRATÉGIQUE DE SÉCURITÉ DE L'INFRASTRUCTURE

Objet : Renforcement de la posture de sécurité pour une infrastructure conteneurisée et automatisée.

Date : 16/10/2025

Auteur : [Votre Nom/Département]

Destinataires : [Comité de Direction / RSSI / Équipe Technique]

1. Introduction et Contexte Notre infrastructure s'appuie sur des technologies modernes de conteneurisation et d'Infrastructure as Code (IaC) via Terraform et Ansible. Cette architecture nous confère une agilité et une résilience significatives. Cependant, elle présente également une surface d'attaque complexe qui nécessite une stratégie de défense multicouche.

Ce document détaille un plan de sécurité holistique articulé autour de trois axes fondamentaux : la **Prévention** pour minimiser les risques, la **Détection** pour identifier les menaces en temps réel, et la **Défense Active** pour neutraliser et analyser les attaquants. Chaque proposition est accompagnée d'un cas d'usage concret applicable à notre environnement.

2. Axe 1 : Stratégies de Prévention L'objectif est de réduire la surface d'attaque en appliquant des contrôles de sécurité stricts en amont.

2.1. Segmentation et Isolation des Services (Micro-segmentation) *

Principe : Appliquer le principe du moindre privilège au niveau du réseau. Par défaut, aucune communication n'est autorisée entre les conteneurs, sauf si elle est explicitement définie. * **Cas Pratique :** * **Outil :** Implémentation de **Calico** ou **Cilium** pour gérer les politiques réseau (Network Policies) dans notre cluster Kubernetes. * **Exemple :** Un conteneur *frontend* est autorisé à communiquer avec un conteneur *API* sur le port TCP/443, mais toute tentative de connexion directe à la base de données depuis le *frontend* est bloquée au niveau réseau par la politique. * **Automatisation :** Les politiques réseau

sont définies sous forme de fichiers YAML et déployées automatiquement via un playbook **Ansible** lors de la création ou de la mise à jour d'un service.

2.2. Gestion Sécurisée des Secrets * **Principe** : Éliminer le stockage de secrets (clés d'API, mots de passe, certificats) en clair dans les fichiers de configuration ou les variables d'environnement. * **Cas Pratique** : * **Outil** : Déploiement d'une solution centralisée comme **HashiCorp Vault**. * **Exemple** : Une application, au démarrage, s'authentifie auprès de Vault en utilisant son identité de service (Service Account Kubernetes). Vault génère alors des identifiants de base de données dynamiques avec une durée de vie limitée (ex: 1 heure), que l'application utilise pour se connecter. * **Automatisation** : **Terraform** provisionne l'infrastructure de Vault et ses politiques d'accès. **Ansible** configure les applications pour qu'elles s'intègrent à Vault.

2.3. Durcissement et Sécurisation de la Chaîne d'Approvisionnement Logicielle (Software Supply Chain) * **Principe** : Garantir l'intégrité et la sécurité des images de conteneurs avant leur déploiement. * **Cas Pratique** : * **Outils** : Intégration dans notre pipeline CI/CD (ex: GitLab CI) des outils suivants : 1. **Trivy** ou **Grype** pour scanner les images et détecter les vulnérabilités (CVEs) connues. 2. **Cosign** (du projet Sigstore) pour signer numériquement les images validées. 3. **Harbor** comme registre privé pour stocker et distribuer les images signées. * **Exemple** : Le pipeline bloque automatiquement la construction si Trivy détecte une vulnérabilité critique. Seules les images scannées et signées peuvent être poussées sur le registre Harbor. L'orchestrateur Kubernetes est configuré pour rejeter tout déploiement d'une image non signée.

3. Axe 2 : Stratégies de Détection L'objectif est d'identifier les activités suspectes ou malveillantes qui auraient pu contourner les mesures de prévention.

3.1. Détection d'Intrusion en Temps Réel (Runtime Security) * **Principe** : Surveiller le comportement des applications à l'intérieur des conteneurs pendant leur exécution pour détecter des anomalies. * **Cas Pratique** : * **Outil** : Déploiement de **Falco** en tant que DaemonSet sur tous les nœuds du cluster. * **Exemple de règle** : Générer une alerte de haute criticité si un shell est lancé (**execve**) dans un conteneur de production qui n'en a pas l'utilité, ou si une écriture est détectée dans un répertoire système sensible comme **/etc**. * **Automatisation** : **Ansible** déploie Falco et un ensemble de règles personnalisées. Les alertes sont agrégées et envoyées vers un système de gestion d'incidents (ex: PagerDuty) ou un canal de communication sécurisé.

3.2. Centralisation et Analyse des Journaux (SIEM) * **Principe** : Collecter, agréger et analyser l'ensemble des journaux d'événements (logs) de l'infrastructure pour la corrélation d'événements et l'investigation numérique (forensics). * **Cas Pratique** : * **Outils** : Mise en place d'une pile **Elasticsearch, Fluentd, Kibana (EFK)** ou d'une solution équivalente. * **Exemple** : Création de tableaux de bord dans Kibana pour visualiser les tentatives de

connexion échouées sur plusieurs services, les pics d’erreurs HTTP 5xx, ou pour corréler une alerte Falco avec les logs applicatifs correspondants. * **Automatisation : Terraform** provisionne les ressources de calcul et de stockage, tandis qu’**Ansible** déploie et configure la pile logicielle.

4. Axe 3 : Stratégies de Défense Active et de Contre-Mesures

L’objectif est de passer d’une posture passive à une posture proactive en piégeant, ralentissant et analysant les attaquants.

4.1. Déploiement de Leurre (Honeypots) * **Principe** : Mettre en place de faux services vulnérables, isolés du réseau de production, conçus pour attirer et tromper les attaquants afin d’étudier leurs méthodes. * **Cas Pratique** : * **Outil** : Déploiement d’un service “honeypot” comme **Kube-honeypot** dans le cluster. * **Exemple** : Un attaquant scannant notre réseau découvre une fausse base de données MongoDB ouverte sans authentification. Toute tentative d’interaction avec cette base est immédiatement signalée, et chaque commande est enregistrée, nous fournissant des renseignements précieux sur ses outils et objectifs.

4.2. Dissémination de Marqueurs (Canary Tokens) * **Principe** : Placer des fichiers, des URL ou des identifiants piégés dans l’infrastructure. Leur utilisation par un attaquant déclenche une alerte silencieuse. * **Cas Pratique** : * **Exemple** : * Insertion d’une fausse clé d’API AWS dans un fichier de configuration. * Création d’un document **Accès_Production_Confidentiel.docx** sur un partage réseau qui, à son ouverture, déclenche une alerte. * Ajout d’une URL unique en commentaire dans le code source. * **Automatisation** : Les “Canary Tokens” sont générés et injectés de manière programmatique via **Ansible** lors des déploiements.

4.3. Neutralisation Automatisée des Menaces (IPS Collaboratif) * **Principe** : Bloquer automatiquement les adresses IP identifiées comme agressives en se basant sur l’analyse comportementale des logs. * **Cas Pratique** : * **Outil** : Utilisation de **CrowdSec**. * **Exemple** : L’agent CrowdSec, en analysant les logs du reverse proxy, détecte une tentative de balayage de vulnérabilités (scan) depuis une IP. Il partage cette information avec le “bouncer” installé sur le pare-feu, qui bloque l’IP pour une durée configurable. La menace est neutralisée avant qu’elle ne puisse aboutir.

4.4. Ralentissement et Désinformation (Tarpitting & Deception) * **Principe** : Rendre l’étape de reconnaissance de l’attaquant extrêmement lente et coûteuse en temps. * **Cas Pratique** : * **Technique** : Configuration du pare-feu en mode “tarpit”. Les ports qui ne correspondent à aucun service légitime ne sont pas fermés, mais répondent avec une lenteur extrême. * **Exemple** : Un scan de ports Nmap, qui prendrait normalement quelques minutes, prendrait alors plusieurs jours. Chaque connexion lente génère une alerte, nous laissant amplement le temps de réagir et de tracer l’origine de l’attaque.

You

donne moi un script pour mettre en place un kube honeypot

Gemini

Bien sûr. Voici un script, sous la forme d'un manifeste YAML Kubernetes, pour déployer un **kube-honeypot**. Ce manifeste est un moyen simple et déclaratif de mettre en place votre piège.

Ce script va créer deux ressources dans votre cluster :

1. Un **Deployment**, qui gère le pod contenant le logiciel honeypot.
2. Un **Service** de type **LoadBalancer**, qui expose le honeypot à l'extérieur de votre cluster pour attirer les attaquants.

Prérequis

Assurez-vous d'avoir un cluster Kubernetes fonctionnel et **kubectl** configuré pour y accéder.

Le Script de Déploiement (Manifeste YAML)

Copiez le contenu suivant dans un fichier nommé **honeypot-deployment.yaml**.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kube-honeypot
  namespace: default # Important : déployez-le dans un namespace isolé, pas "default" en pr
  labels:
    app: kube-honeypot
spec:
  replicas: 1 # On déploie une seule instance du piège.
  selector:
    matchLabels:
      app: kube-honeypot
  template:
    metadata:
      labels:
        app: kube-honeypot
    spec:
      containers:
```

```

- name: honeypot-container
  image: dkube/kube-honeypot:1.3.0 # L'image officielle du honeypot.
  # --- Personnalisez vos pièges ici ---
  # Active les leurres souhaités. Chaque leurre écoute sur son port par défaut.
  args:
    - "--honeypots=http,elasticsearch,redis,memcached"
  ports:
    - containerPort: 80 # HTTP
    - containerPort: 9200 # Elasticsearch
    - containerPort: 6379 # Redis
    - containerPort: 11211 # Memcached
  resources:
    limits:
      cpu: "100m"
      memory: "128Mi"
    requests:
      cpu: "50m"
      memory: "64Mi"
---
apiVersion: v1
kind: Service
metadata:
  name: honeypot-service
  namespace: default # Doit être dans le même namespace que le Deployment.
  annotations:
    # Pour certains fournisseurs cloud, cette annotation est nécessaire pour obtenir une IP
    service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
spec:
  type: LoadBalancer # Rend le honeypot accessible de l'extérieur. ATTENTION !
  selector:
    app: kube-honeypot # Cible les pods avec le label app=kube-honeypot.
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 80
    - name: elasticsearch
      protocol: TCP
      port: 9200
      targetPort: 9200
    - name: redis
      protocol: TCP
      port: 6379
      targetPort: 6379
    - name: memcached
      protocol: TCP

```

```
port: 11211
targetPort: 11211
```

Mise en Place (Déploiement)

1. **Enregistrez le fichier** : Sauvegardez le code ci-dessus dans `honeypot-deployment.yaml`.
2. **Appliquez le manifeste** : Ouvrez un terminal et exécutez la commande suivante :

```
kubectl apply -f honeypot-deployment.yaml
```

Cette commande va demander à Kubernetes de créer les ressources décrites dans le fichier.

Comment Vérifier et Voir les Alertes

1. **Vérifiez le déploiement** : Attendez une minute, puis vérifiez que le pod est bien en cours d'exécution.

```
kubectl get pods -l app=kube-honeypot
```

Vous devriez voir un pod avec le statut **Running**.

2. **Trouvez l'adresse IP du piège** : Le service de type **LoadBalancer** va provisionner une adresse IP publique. Trouvez-la avec cette commande :

```
kubectl get service honeypot-service
```

Repérez l'adresse dans la colonne **EXTERNAL-IP**. Cela peut prendre quelques minutes pour apparaître.

3. **Simulez une attaque** : Utilisez l'adresse IP externe que vous venez de trouver pour interagir avec le honeypot.

- **Test du port HTTP** : `bash curl http://<VOTRE_IP_EXTERNE>`
- **Test du port Redis** : `bash # Si vous avez redis-cli installé redis-cli -h <VOTRE_IP_EXTERNE> info`

4. **Consultez les logs d'attaque** : C'est ici que la magie opère. Toutes les tentatives de connexion sont enregistrées dans les logs du pod. Pour les consulter :

```
kubectl logs -f deployment/kube-honeypot
```

Vous verrez des sorties JSON détaillées pour chaque tentative de connexion, comme ceci :

```
{"Honeypot":"http","RemoteAddr":"123.45.67.89:54321","Time":"2025-10-16T14:30:00Z","Data":{}}, {"Honeypot":"redis","RemoteAddr":"123.45.67.89:54322","Time":"2025-10-16T14:31:15Z","Data":{}}
```

Chaque ligne vous donne l'adresse IP de l'attaquant (**RemoteAddr**), le service visé (**Honeypot**) et les données de la tentative (**Data**).

Points d'Attention et Sécurité

- **Isolation Réseau** : **NE JAMAIS** déployer un honeypot dans un namespace de production. Créez un namespace dédié (ex: `honeypot-ns`). Idéalement, appliquez une **NetworkPolicy** stricte pour vous assurer que ce pod ne peut **JAMAIS** initier de connexions vers vos services de production. Il doit être une impasse totale.
- **Coûts** : Un service de type **LoadBalancer** engendre des coûts chez les fournisseurs de cloud (AWS, GCP, Azure).
- **Personnalisation** : Vous pouvez facilement ajouter d'autres types de leurres en modifiant la ligne **args** du manifeste. La liste des honeypots disponibles est dans la documentation du projet `kube-honeypot`.

Ce script vous donne un point de départ fonctionnel pour intégrer la déception et la défense active directement dans votre infrastructure Kubernetes.

You

fait moi un script pour mettre en place le scénario suivant Création d'un document `Accès_Production_Confidentiel.docx` sur un partage réseau qui, à son ouverture, déclenche une alerte.

Gemini

Absolument. Mettre en place ce scénario est un excellent exemple de défense active utilisant des **Canary Tokens**.

Vous ne pouvez pas créer ce type de document avec un simple script `bash` seul, car le mécanisme d'alerte doit être généré par un service spécialisé. Le "script" consiste donc en deux étapes :

1. **Générer** le document piégé via un service de Canary Tokens.
2. **Déployer** le document sur le partage réseau avec un script.

Voici le guide complet et le script pour le faire.

Étape 1 : Création du Document Piégé

Nous allons utiliser le service gratuit et très réputé **Canarytokens.org** pour générer notre document Word.

1. **Rendez-vous sur le site** : <https://canarytokens.org>
2. **Configurez votre token** :
 - **Your email address** : Saisissez l'adresse e-mail où vous souhaitez recevoir l'alerte (par exemple, `alertes-securite@votre-ecole.fr`).
 - **A reminder for this token** : C'est la partie la plus importante pour le contexte. Mettez une description très claire. Exemple : Document Word 'Accès_Production' sur le partage //SRV-FICHIERS/Commun.
 - Dans la liste des tokens, sélectionnez **MS Word**.
3. **Cliquez sur "Create my Canarytoken"**.
4. **Téléchargez le document** : Le site va générer et vous faire télécharger un fichier `document.docx`. **Ce fichier est votre piège.**

Étape 2 : Le Script de Déploiement (Bash)

Maintenant que vous avez votre fichier piégé, il faut le renommer de manière attractive et le placer discrètement sur votre partage réseau.

Ce script Bash automatise le renommage et la copie du fichier.

1. Placez le fichier `document.docx` que vous avez téléchargé dans un répertoire, par exemple `/tmp/`.
2. Créez un fichier nommé `deploy_canary_doc.sh` et copiez-y le contenu suivant :

```
#!/bin/bash
```

```
# --- VARIABLES À CONFIGURER ---
```

```
# Chemin vers le fichier .docx téléchargé depuis Canarytokens.org
```

```
SOURCE_FILE="/tmp/document.docx"
```

```
# Chemin de destination sur le partage réseau (assurez-vous qu'il est monté sur votre système)
```

```
# Exemple pour un partage monté localement :
```

```
DESTINATION_SHARE="/mnt/partage_commun/Documents_Importants/"
```

```
# Le nom de fichier final, conçu pour être attractif
```

```
NEW_FILENAME="Accès_Production_Confidentiel.docx"
```

```

# --- SCRIPT ---

echo "--- Déploiement du Canary Token Document ---"

# Vérifier si le fichier source existe
if [ ! -f "$SOURCE_FILE" ]; then
    echo "ERREUR : Le fichier source '$SOURCE_FILE' n'a pas été trouvé."
    exit 1
fi

# Vérifier si le répertoire de destination existe
if [ ! -d "$DESTINATION_SHARE" ]; then
    echo "ERREUR : Le répertoire de destination '$DESTINATION_SHARE' n'existe pas ou n'est p
    exit 1
fi

# Copier et renommer le fichier
cp "$SOURCE_FILE" "$DESTINATION_SHARE$NEW_FILENAME"

# Vérifier si la copie a réussi
if [ $? -eq 0 ]; then
    echo " Succès ! Le document piégé a été placé ici : $DESTINATION_SHARE$NEW_FILENAME"
else
    echo " ÉCHEC : Impossible de copier le fichier vers la destination."
    exit 1
fi

echo "--- Opération terminée ---"

```

3. Rendez le script exécutable :

```
chmod +x deploy_canary_doc.sh
```

4. Exécutez le script :

```
./deploy_canary_doc.sh
```

Le script va copier le fichier au bon endroit avec le nom spécifié. Le piège est maintenant en place !

Étape 3 : Que se passe-t-il lors de l'ouverture ?

Dès que quelqu'un (un utilisateur curieux ou un attaquant) ouvrira le fichier `Accès_Production_Confidentiel.docx`, vous recevrez une alerte par e-mail quasi instantanément.

L'alerte ressemblera à ceci :

Subject: Canarytoken triggered

You have a notification for your Canarytoken.

Reminder: Document Word ‘Accès_Production’ sur le partage
//SRV-FICHIERS/Commun **Token:** xxxxxxxxxxxxxxxx.docx

The token was triggered by **81.240.XX.XX** (France Telecom) from
a Microsoft Word client.

Manage this token here: [lien de gestion]

Vous obtenez l'**adresse IP** de la personne qui a ouvert le document et la confirmation que c'était bien via Microsoft Word.

Comment Ça Marche Techniquement ?

Le document .docx généré n'est pas un simple fichier texte. Il contient une **ressource externe invisible**, généralement une image de 1x1 pixel, hébergée sur les serveurs de Canarytokens.org avec une URL unique.

Lorsque Microsoft Word (ou tout autre lecteur compatible) ouvre le document, il essaie de charger toutes les ressources externes, y compris cette image invisible. En tentant de la télécharger, il fait une requête DNS et HTTP vers l'URL unique. Le serveur de Canarytokens détecte cette requête, sait que votre piège a été déclenché, et vous envoie l'alerte par e-mail. C'est simple, discret et très efficace.

You**DOCUMENTATION STRATÉGIQUE DE SÉCURITÉ DE L'INFRASTRUCTURE**

Objet : Renforcement de la posture de sécurité pour une infrastructure conteneurisée et automatisée. Date : 16/10/2025 Auteur : [Votre Nom/Département] Destinataires : [Comité de Direction / RSSI / Équipe Technique] 1. Introduction et Contexte Notre infrastructure s'appuie sur des technologies modernes de conteneurisation et d'Infrastructure as Code (IaC) via Terraform et Ansible. Cette architecture nous confère une agilité et une résilience significatives. Cependant, elle présente également une surface d'attaque complexe qui nécessite une stratégie de défense multicouche. Ce document détaille un plan de sécurité holistique articulé autour de trois axes fondamentaux : la Prévention pour minimiser les risques, la Détection pour identifier les menaces en temps réel, et la Défense Active pour neutraliser et analyser les attaquants. Chaque proposition est accompagnée d'un cas d'usage concret applicable à notre environnement. 2. Axe 1 : Stratégies de Prévention L'objectif est de réduire la surface d'attaque en appliquant des contrôles de sécurité stricts en amont. 2.1. Segmentation et Isolation des Services

(Micro-segmentation) Principe : Appliquer le principe du moindre privilège au niveau du réseau. Par défaut, aucune communication n'est autorisée entre les conteneurs, sauf si elle est explicitement définie. Cas Pratique : Outil : Implémentation de Calico ou Cilium pour gérer les politiques réseau (Network Policies) dans notre cluster Kubernetes. Exemple : Un conteneur frontend est autorisé à communiquer avec un conteneur API sur le port TCP/443, mais toute tentative de connexion directe à la base de données depuis le frontend est bloquée au niveau réseau par la politique. Automatisation : Les politiques réseau sont définies sous forme de fichiers YAML et déployées automatiquement via un playbook Ansible lors de la création ou de la mise à jour d'un service.

2.2. Gestion Sécurisée des Secrets Principe : Éliminer le stockage de secrets (clés d'API, mots de passe, certificats) en clair dans les fichiers de configuration ou les variables d'environnement. Cas Pratique : Outil : Déploiement d'une solution centralisée comme HashiCorp Vault. Exemple : Une application, au démarrage, s'authentifie auprès de Vault en utilisant son identité de service (Service Account Kubernetes). Vault génère alors des identifiants de base de données dynamiques avec une durée de vie limitée (ex: 1 heure), que l'application utilise pour se connecter. Automatisation : Terraform provisionne l'infrastructure de Vault et ses politiques d'accès. Ansible configure les applications pour qu'elles s'intègrent à Vault.

2.3. Durcissement et Sécurisation de la Chaîne d'Approvisionnement Logicielle (Software Supply Chain) Principe : Garantir l'intégrité et la sécurité des images de conteneurs avant leur déploiement. Cas Pratique : Outils : Intégration dans notre pipeline CI/CD (ex: GitLab CI) des outils suivants : Trivy ou Gype pour scanner les images et détecter les vulnérabilités (CVEs) connues. Cosign (du projet Sigstore) pour signer numériquement les images validées. Harbor comme registre privé pour stocker et distribuer les images signées. Exemple : Le pipeline bloque automatiquement la construction si Trivy détecte une vulnérabilité critique. Seules les images scannées et signées peuvent être poussées sur le registre Harbor. L'orchestrateur Kubernetes est configuré pour rejeter tout déploiement d'une image non signée.

3. Axe 2 : Stratégies de Détection L'objectif est d'identifier les activités suspectes ou malveillantes qui auraient pu contourner les mesures de prévention.

3.1. Détection d'Intrusion en Temps Réel (Runtime Security) Principe : Surveiller le comportement des applications à l'intérieur des conteneurs pendant leur exécution pour détecter des anomalies. Cas Pratique : Outil : Déploiement de Falco en tant que DaemonSet sur tous les nœuds du cluster. Exemple de règle : Générer une alerte de haute criticité si un shell est lancé (execve) dans un conteneur de production qui n'en a pas l'utilité, ou si une écriture est détectée dans un répertoire système sensible comme /etc. Automatisation : Ansible déploie Falco et un ensemble de règles personnalisées. Les alertes sont agrégées et envoyées vers un système de gestion d'incidents (ex: PagerDuty) ou un canal de communication sécurisé.

3.2. Centralisation et Analyse des Journaux (SIEM) Principe : Collecter, agréger et analyser l'ensemble des journaux d'événements (logs) de l'infrastructure pour la corrélation d'événements et l'investigation numérique (forensics). Cas Pratique : Outils : Mise en place d'une pile Elasticsearch, Fluentd, Kibana

(EFK) ou d’une solution équivalente. Exemple : Création de tableaux de bord dans Kibana pour visualiser les tentatives de connexion échouées sur plusieurs services, les pics d’erreurs HTTP 5xx, ou pour corréler une alerte Falco avec les logs applicatifs correspondants. Automatisation : Terraform provisionne les ressources de calcul et de stockage, tandis qu’Ansible déploie et configure la pile logicielle.

4. Axe 3 : Stratégies de Défense Active et de Contre-Mesures L’objectif est de passer d’une posture passive à une posture proactive en piégeant, ralentissant et analysant les attaquants.

4.1. Déploiement de Leurre (Honeypots) Principe : Mettre en place de faux services vulnérables, isolés du réseau de production, conçus pour attirer et tromper les attaquants afin d’étudier leurs méthodes. Cas Pratique : Outil : Déploiement d’un service “honeypot” comme Kube-honeypot dans le cluster. Exemple : Un attaquant scannant notre réseau découvre une fausse base de données MongoDB ouverte sans authentification. Toute tentative d’interaction avec cette base est immédiatement signalée, et chaque commande est enregistrée, nous fournissant des renseignements précieux sur ses outils et objectifs.

4.2. Dissémination de Marqueurs (Canary Tokens) Principe : Placer des fichiers, des URL ou des identifiants piégés dans l’infrastructure. Leur utilisation par un attaquant déclenche une alerte silencieuse. Cas Pratique : Exemple : Insertion d’une fausse clé d’API AWS dans un fichier de configuration. Création d’un document `Accès_Production_Confidentiel.docx` sur un partage réseau qui, à son ouverture, déclenche une alerte. Ajout d’une URL unique en commentaire dans le code source. Automatisation : Les “Canary Tokens” sont générés et injectés de manière programmatique via Ansible lors des déploiements.

4.3. Neutralisation Automatisée des Menaces (IPS Collaboratif) Principe : Bloquer automatiquement les adresses IP identifiées comme agressives en se basant sur l’analyse comportementale des logs. Cas Pratique : Outil : Utilisation de CrowdSec. Exemple : L’agent CrowdSec, en analysant les logs du reverse proxy, détecte une tentative de balayage de vulnérabilités (scan) depuis une IP. Il partage cette information avec le “bouncer” installé sur le pare-feu, qui bloque l’IP pour une durée configurable. La menace est neutralisée avant qu’elle ne puisse aboutir.

4.4. Ralentissement et Désinformation (Tarpitting & Deception) Principe : Rendre l’étape de reconnaissance de l’attaquant extrêmement lente et coûteuse en temps. Cas Pratique : Technique : Configuration du pare-feu en mode “tarpit”. Les ports qui ne correspondent à aucun service légitime ne sont pas fermés, mais répondent avec une lenteur extrême. Exemple : Un scan de ports Nmap, qui prendrait normalement quelques minutes, prendrait alors plusieurs jours. Chaque connexion lente génère une alerte, nous laissant amplement le temps de réagir et de tracer l’origine de l’attaque. pour chaque solution proposé donne moi un tutoriel pour l’installation et la place dans l’infra

Gemini

Absolument. Voici les tutoriels d’installation et de positionnement pour chaque solution proposée dans votre documentation stratégique de sécurité.

Ce guide est conçu pour être un complément technique, fournissant les étapes pratiques pour mettre en œuvre la stratégie.

GUIDE D'IMPLÉMENTATION TECHNIQUE

Ce document détaille les procédures d'installation et de configuration pour chaque outil de sécurité mentionné dans la stratégie de défense. Chaque section explique le rôle de l'outil, sa place dans l'infrastructure et fournit un guide étape par étape pour son déploiement.

AXE 1 : TUTORIELS DE PRÉVENTION

1. Calico pour la Micro-segmentation

- **Rôle et Place dans l'Infrastructure : Calico** s'intègre directement à votre cluster Kubernetes en tant que CNI (Container Network Interface). Il se place au cœur du réseau de vos conteneurs et agit comme un pare-feu distribué, appliquant des règles de communication (**NetworkPolicy**) entre chaque pod. Il est essentiel pour implémenter une architecture **Zero Trust** au sein du cluster.
- **Tutoriel d'Installation et Configuration** : L'installation la plus simple se fait via l'opérateur Calico.

1. Installer l'opérateur Calico :

```
kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.28.0/ma
```

- #### 2. Appliquer la configuration par défaut :
- Cette étape installe Calico avec les réglages standards.

```
kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.28.0/ma
```

- #### 3. Vérifier l'installation :
- Attendez quelques minutes que tous les pods Calico soient lancés.

```
kubectl get pods -n calico-system
```

- **Exemple Concret : Isoler la base de données** Voici un playbook **Ansible** qui déploie une **NetworkPolicy** pour s'assurer que seuls les pods de l'API (**app: api-serdaigle**) peuvent parler à la base de données (**app: db-poufsouffle**) sur le port 5432.

Créez un fichier `deploy_db_policy.yml` :

```

- name: Appliquer la politique réseau de la base de données
  hosts: localhost
  tasks:
    - name: Déployer la NetworkPolicy
      kubernetes.core.k8s:
        state: present
        definition:
          apiVersion: networking.k8s.io/v1
          kind: NetworkPolicy
          metadata:
            name: allow-api-to-db
            namespace: production
          spec:
            podSelector:
              matchLabels:
                app: db-poufsouffle
            policyTypes:
              - Ingress
            ingress:
              - from:
                  - podSelector:
                      matchLabels:
                        app: api-serdaigle
            ports:
              - protocol: TCP
                port: 5432

```

Exécutez-le avec `ansible-playbook deploy_db_policy.yml`. Toute autre tentative de connexion à la base de données sera désormais bloquée.

2. HashiCorp Vault pour la Gestion des Secrets

- **Rôle et Place dans l'Infrastructure : Vault** est un service centralisé externe à votre cluster Kubernetes (bien qu'il puisse aussi y être déployé). Il agit comme le coffre-fort central de votre infrastructure. Les applications, au lieu de stocker des secrets, obtiennent un "jeton d'identité" au démarrage et l'utilisent pour demander à Vault les secrets dont elles ont besoin.
- **Tutoriel d'Installation et Configuration (simplifié) :**

1. **Provisionnement avec Terraform :** Utilisez Terraform pour créer une VM dédiée pour Vault.

```

resource "google_compute_instance" "vault_server" {
  name = "vault-server-prod"

```

```

machine_type = "e2-medium"
zone         = "europe-west1-b"
# ... configuration du boot disk, réseau, etc.
}

```

2. **Installation avec Ansible :** Utilisez un playbook pour installer et configurer Vault sur la VM.

```

- name: Installer et configurer Vault
  hosts: vault_servers
  tasks:
    - name: Installer Vault
      # ... tâches pour télécharger et installer le binaire de Vault
    - name: Configurer Vault
      # ... tâches pour créer le fichier de configuration et démarrer le service
    - name: Initialiser et desceller Vault
      # ... tâches pour exécuter `vault operator init` et stocker les clés

```

- **Exemple Concret : Accès dynamique à la base de données**
L'application dans Kubernetes va utiliser son **Service Account Token** pour s'authentifier auprès de Vault.

1. **Configuration dans Vault :** Vous configurez Vault pour qu'il fasse confiance aux tokens des Service Accounts de votre cluster.

```

# Activer l'authentification Kubernetes dans Vault
vault auth enable kubernetes

```

```

# Configurer Vault avec les détails de votre cluster
vault write auth/kubernetes/config token_reviewer_jwt="..." kubernetes_host="..."

```

```

# Créer un rôle qui lie un Service Account (ex: 'api-serdaigle') à une politique Vault
vault write auth/kubernetes/role/api-serdaigle \
  bound_service_account_names=api-serdaigle \
  bound_service_account_namespaces=production \
  policies=db-access-policy \
  ttl=24h

```

2. **Côté application :** L'application, en démarrant, lit son propre token de service, l'envoie à Vault, reçoit un token Vault en retour, puis l'utilise pour demander un mot de passe de base de données temporaire. Ce processus est souvent géré par un "sidecar injector" pour être transparent pour les développeurs.

3. Sécurisation de la CI/CD (Trivy, Cosign, Harbor)

- **Rôle et Place dans l'Infrastructure** : Ces outils s'intègrent directement dans votre **pipeline CI/CD** (ex: GitLab CI). Ils agissent comme des points de contrôle de qualité et de sécurité. **Trivy** est le gardien qui inspecte les "ingrédients" (les paquets), **Cosign** est le sceau de cire qui garantit l'authenticité, et **Harbor** est le coffre-fort où les images validées sont stockées.
- **Tutoriel d'Installation et Configuration** : Ces outils sont utilisés comme des étapes dans votre fichier `.gitlab-ci.yml`.
 1. **Harbor** : Installez Harbor sur une VM ou dans Kubernetes. C'est votre registre d'images privé.
 2. **GitLab CI** : Configurez des variables secrètes dans GitLab (`HARBOR_USER`, `HARBOR_PASSWORD`) pour permettre au pipeline de se connecter à Harbor.
- **Exemple Concret : Pipeline CI/CD Sécurisé** Voici à quoi pourrait ressembler votre fichier `.gitlab-ci.yml` :

```
stages:
  - build
  - scan
  - sign_and_push

build_image:
  stage: build
  script:
    - docker build -t $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA .

scan_image:
  stage: scan
  image: aquasec/trivy:latest
  script:
    # Bloque le pipeline si une vulnérabilité CRITIQUE est trouvée.
    - trivy image --exit-code 1 --severity CRITICAL $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA

sign_and_push_image:
  stage: sign_and_push
  image: gcr.io/projectsigstore/cosign:v1.3.1
  script:
    # Connexion à votre registre Harbor
    - echo $HARBOR_PASSWORD | docker login harbor.votre-ecole.fr -u $HARBOR_USER --password-stdin
    # Pousser l'image
    - docker push harbor.votre-ecole.fr/production/$CI_PROJECT_NAME:$CI_COMMIT_SHA
    # Signer l'image poussée (Cosign utilise les variables d'environnement pour l'authentification)
    - cosign sign harbor.votre-ecole.fr/production/$CI_PROJECT_NAME:$CI_COMMIT_SHA
```

AXE 2 : TUTORIELS DE DÉTECTION

4. Falco pour la Détection d’Intrusion en Temps Réel

- **Rôle et Place dans l’Infrastructure :** Falco se déploie en tant que **DaemonSet** sur chaque nœud de votre cluster Kubernetes. Il se branche directement sur le noyau du système d’exploitation pour surveiller les appels système. Il est votre “œil magique” qui voit tout ce qui se passe à un niveau très bas et vous alerte en cas de comportement suspect (lancement d’un shell, modification de fichiers critiques, etc.).
- **Tutoriel d’Installation et Configuration :** L’installation via Helm est la plus simple.

1. **Ajouter le dépôt Helm de Falco :**

```
helm repo add falcosecurity https://falcosecurity.github.io/charts
helm repo update
```

2. **Installer Falco :**

```
helm install falco falcosecurity/falco \
  --namespace falco --create-namespace \
  --set falcosidekick.enabled=true \
  --set falcosidekick.webui.enabled=true
```

(Falcosidekick est un outil pratique pour envoyer les alertes de Falco vers Slack, PagerDuty, etc.)

- **Exemple Concret : Détecter l’écriture dans /etc** Les règles de Falco sont dans un fichier `falco_rules.yaml`. Vous pouvez ajouter des règles personnalisées.

1. **Créer un fichier de règles personnalisées** `custom-rules.yaml`:

```
yaml    - rule: Write below etc      desc: an attempt
to write to /etc file      condition: open_write and
fd.name contains /etc/ and not proc_name in (user_known_write_etc_procs)
output: "File below /etc opened for writing (user=%user.name
command=%proc.cmdline file=%fd.name)"      priority:
ERROR
```

2. **Mettre à jour votre déploiement Helm** pour inclure ces règles.

Maintenant, si quelqu’un lance `echo "test" >> /etc/hosts` dans un conteneur, Falco générera immédiatement une alerte.

5. Pile EFK pour la Centralisation des Logs

- **Rôle et Place dans l'Infrastructure** : La pile EFK (Elasticsearch, Fluentd, Kibana) est une solution de journalisation centralisée.
 - **Fluentd** est déployé en **DaemonSet** sur chaque nœud pour collecter les logs de tous les conteneurs.
 - **Elasticsearch** est la base de données, déployée en tant que **StatefulSet** dans le cluster, où tous les logs sont stockés et indexés.
 - **Kibana** est l'interface web (**Deployment + Service**) qui vous permet de rechercher, visualiser et créer des tableaux de bord à partir des logs.
- **Tutoriel d'Installation et Configuration** : Utiliser des opérateurs ou des charts Helm est recommandé.
 1. **Installer Elasticsearch** : Utilisez le chart Helm d'Elastic.

```
helm repo add elastic https://helm.elastic.co
helm install elasticsearch elastic/elasticsearch --namespace logging
```
 2. **Installer Fluentd** : Utilisez un chart Helm configuré pour envoyer les logs à Elasticsearch. Fluentd montera les répertoires de logs des conteneurs de l'hôte (/var/log/pods).

```
helm install fluentd fluent/fluentd --namespace logging \
  --set elasticsearch.host="elasticsearch-master"
```
 3. **Installer Kibana** :

```
helm install kibana elastic/kibana --namespace logging
```
- **Exemple Concret : Créer un tableau de bord des erreurs 4xx**
Une fois la pile déployée, accédez à l'interface de Kibana.
 1. Allez dans la section "Discover" pour voir les logs arriver.
 2. Filtrez sur `kubernetes.namespace_name: "production"` et `http_status_code: 4*`.
 3. Allez dans "Dashboard", créez un nouveau tableau de bord et ajoutez une visualisation (ex: un graphique en barres) montrant le nombre d'erreurs 403 par nom de pod. Vous avez maintenant une vue en temps réel des tentatives d'accès non autorisé.

AXE 3 : TUTORIELS DE DÉFENSE ACTIVE

6. Kube-Honeypot pour les Leurres

- **Rôle et Place dans l'Infrastructure : Kube-honeypot** est un **leurre**. Il est déployé comme un service normal (**Deployment** + **Service**) dans votre cluster, de préférence dans un namespace **isolé**. Son but est d'être découvert par un attaquant lors d'une phase de reconnaissance. En exposant de faux services (fausse base de données, fausse API), il vous alerte de la présence d'un intrus et vous permet d'analyser ses actions sans risque pour votre production.
- **Tutoriel d'Installation et Configuration** : Cela se fait avec un simple manifeste YAML (comme fourni précédemment).

1. Créer un namespace dédié :

```
kubectl create namespace honeypot
```

2. Déployer le manifeste honeypot-deployment.yaml dans ce namespace :

(Utilisez le manifeste fourni dans la réponse précédente, en changeant `namespace: default` par `namespace: honeypot`).

```
kubectl apply -f honeypot-deployment.yaml -n honeypot
```

- **Exemple Concret : Isoler le Honeypot** La sécurité du honeypot réside dans son **isolement**. Appliquez une **NetworkPolicy** qui l'empêche de contacter quoi que ce soit d'autre dans le cluster.

Créez `honeypot-deny-all.yaml`:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all-egress
  namespace: honeypot
spec:
  podSelector:
    matchLabels:
      app: kube-honeypot
  policyTypes:
    - Egress # Appliquer uniquement aux sorties
  egress: [] # Une liste vide bloque tout trafic sortant
```

Appliquez-le avec `kubectl apply -f honeypot-deny-all.yaml`. Votre honeypot peut être contacté, mais il ne peut contacter personne.

7. Canary Tokens pour les Marqueurs

- **Rôle et Place dans l'Infrastructure** : Les **Canary Tokens** sont des “fichiers espions” que vous dispersez dans votre infrastructure : dans vos

conteneurs, sur vos partages de fichiers, dans votre code source. Ils sont passifs. Leur valeur se révèle lorsqu'un attaquant les trouve et interagit avec (en ouvrant un document, en utilisant une fausse clé API...). Ils sont un excellent moyen de détecter une exfiltration de données ou une reconnaissance interne.

- **Tutoriel d'Installation et Configuration :** Le "script" est un playbook Ansible qui place les tokens générés.

1. **Générez vos tokens** sur Canarytokens.org (un token par type : document Word, clé AWS, URL).
2. **Stockez les tokens** (le fichier .docx, la chaîne de la clé AWS) dans un répertoire sécurisé de votre projet Ansible.

- **Exemple Concret : Playbook Ansible de dissémination**

```
- name: Disséminer des Canary Tokens
  hosts: kubernetes_nodes
  tasks:
    - name: Placer un faux fichier de configuration AWS
      copy:
        dest: "/etc/secrets/aws_credentials.conf"
        content: |
          [default]
          aws_access_key_id = AAAAAA... # Votre fausse clé AWS ici
          aws_secret_access_key = XXXXXX...

    - name: Placer le document Word piégé sur un partage
      copy:
        src: files/Accès_Production_Confidentiel.docx
        dest: /mnt/partage_commun/Direction/
```

8. CrowdSec pour la Neutralisation Automatisée

- **Rôle et Place dans l'Infrastructure :** CrowdSec est un système de détection et de blocage collaboratif.
 - L'**Agent CrowdSec** s'installe au plus près de vos services exposés (sur vos nœuds Ingress, vos serveurs web). Il lit les logs en temps réel pour détecter les comportements malveillants (scans, brute force, etc.).
 - Le **Bouncer** s'installe sur votre point de contrôle (pare-feu, reverse proxy). Il reçoit les ordres de blocage de l'agent et les applique. La force de CrowdSec est sa **communauté** : si un attaquant est détecté par un utilisateur, son IP est partagée avec tous les autres, offrant une protection proactive.

- **Tutoriel d'Installation et Configuration :**

1. **Installer l'Agent CrowdSec** sur un nœud qui exécute votre Ingress Controller :

```
curl -s https://packagecloud.io/install/repositories/crowdsec/crowdsec/script.deb.sh  
sudo apt-get install crowdsec
```

2. **Installer un Bouncer** (ex: pour Nginx) :

```
sudo apt-get install crowdsec-nginx-bouncer
```

3. **Vérifier le statut** : L'outil en ligne de commande `cscli` est votre ami.

```
sudo cscli metrics  
sudo cscli decisions list
```

- **Exemple Concret : Blocage d'un scan de vulnérabilité** Un attaquant lance un scanner comme Nikto contre votre site web.

1. L'**Agent CrowdSec** analyse les logs de Nginx, voit des centaines de requêtes suspectes en quelques secondes et identifie ce comportement comme un "http-crawl-non-common-path".
2. L'Agent crée une **décision** de bannir l'IP de l'attaquant.
3. Le **Bouncer Nginx** reçoit cette décision et ajoute une règle dans Nginx pour renvoyer une erreur 403 **Forbidden** à cette IP. L'attaque est bloquée en quelques secondes, et l'IP de l'attaquant est partagée avec la communauté.

9. Tarpitting pour la Désinformation

- **Rôle et Place dans l'Infrastructure** : Le **Tarpitting** (pot de goudron) n'est pas un outil mais une **technique de configuration de pare-feu**. Elle se place à la **périphérie de votre réseau** (sur votre pare-feu Edge, avant même votre Load Balancer Cloud). Son but est de ralentir considérablement les attaquants en phase de reconnaissance, rendant leurs scans de ports interminables et coûteux.
- **Tutoriel d'Installation et Configuration** : Cela se fait avec `iptables` sur un pare-feu Linux.

1. **Créez une nouvelle chaîne iptables** nommée TARPIT.

```
sudo iptables -N TARPIT
```

2. **Configurez la chaîne** pour qu'elle piège les nouvelles connexions TCP en réduisant la taille de la fenêtre à zéro, forçant le client à attendre indéfiniment.

```
sudo iptables -A TARPIT -p tcp -j TCPMSS --set-mss 1
sudo iptables -A TARPIT -j REJECT --reject-with tcp-reset
```

(Note : des implémentations plus avancées existent)

- **Exemple Concret : Piéger tous les ports non utilisés** Imaginez que votre service web n'écoute que sur les ports 80 et 443. Vous pouvez rediriger tout le reste vers votre piège.

```
# Autoriser le trafic établi
```

```
sudo iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
```

```
# Autoriser le trafic légitime (SSH, HTTP, HTTPS)
```

```
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

```
sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
```

```
sudo iptables -A INPUT -p tcp --dport 443 -j ACCEPT
```

```
# Envoyer tout le reste dans le pot de goudron
```

```
sudo iptables -A INPUT -p tcp -j TARPIT
```

Un `nmap` sur votre infrastructure prendra maintenant des heures ou des jours au lieu de minutes, chaque connexion lente étant une opportunité de détection pour vous.