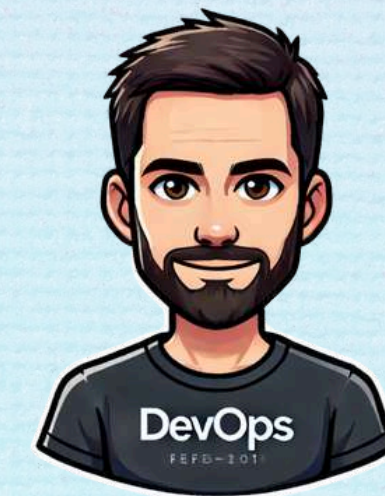


PRINCIPES DE SÉCURITÉ KUBERNETES

APPROCHES CLÉS POUR SÉCURISER VOTRE CLUSTER



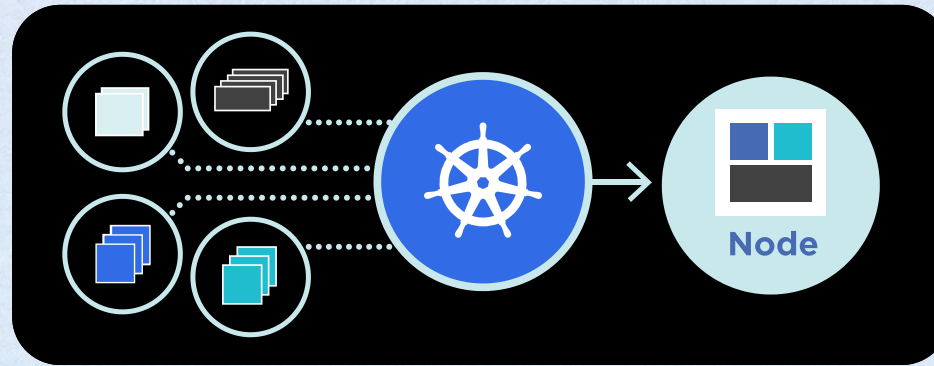
OFFRIR UNE VISION CLAIRE, STRUCTURÉE
ET ACCESSIBLE DES BONNES PRATIQUES
POUR SÉCURISER EFFICACEMENT UN
ENVIRONNEMENT KUBERNETES.



PRÉSENTÉ PAR: TYLER

KUBERNETES EN BREF

KUBERNETES (K8S) EST UN SYSTÈME OPEN-SOURCE PERMETTANT D'AUTOMATISER LE DÉPLOIEMENT, LA MISE À L'ÉCHELLE ET LA GESTION DES APPLICATIONS CONTENEURISÉES.



Documentation Kubernetes



Portabilité : fonctionne sur tous types d'infrastructures (cloud, on-premise)

Résilience : redémarre automatiquement les services qui tombent

Scalabilité : adapte les ressources selon la charge

Automatisation : réduit les erreurs humaines et les tâches répétitives

Mises à jour régulières : une nouvelle version tous les 3 mois

Gérée par la CNCF (Cloud Native Computing Foundation), avec des milliers de contributeurs dans le monde

+ Documentation de qualité, communauté d'entraide, conférences (KubeCon), blogs techniques, forums actifs



Architecture distribuée :

- Plusieurs composants interconnectés.
- Fonctionne en cluster (compréhension d'interaction.)

Modèle déclaratif & abstrait

- connaissance de l'état souhaité.
- Compréhension du fonctionnement des objets & relations.

Configuration en YAML

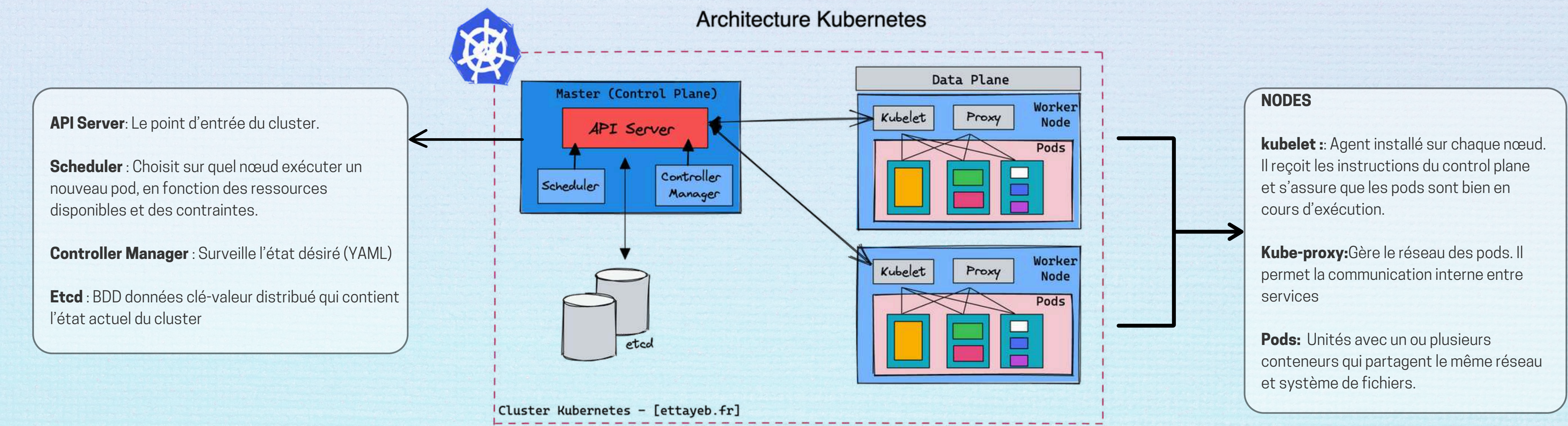
- Beaucoup de fichiers verbeux, complexes, peu lisibles.

Sécurité :

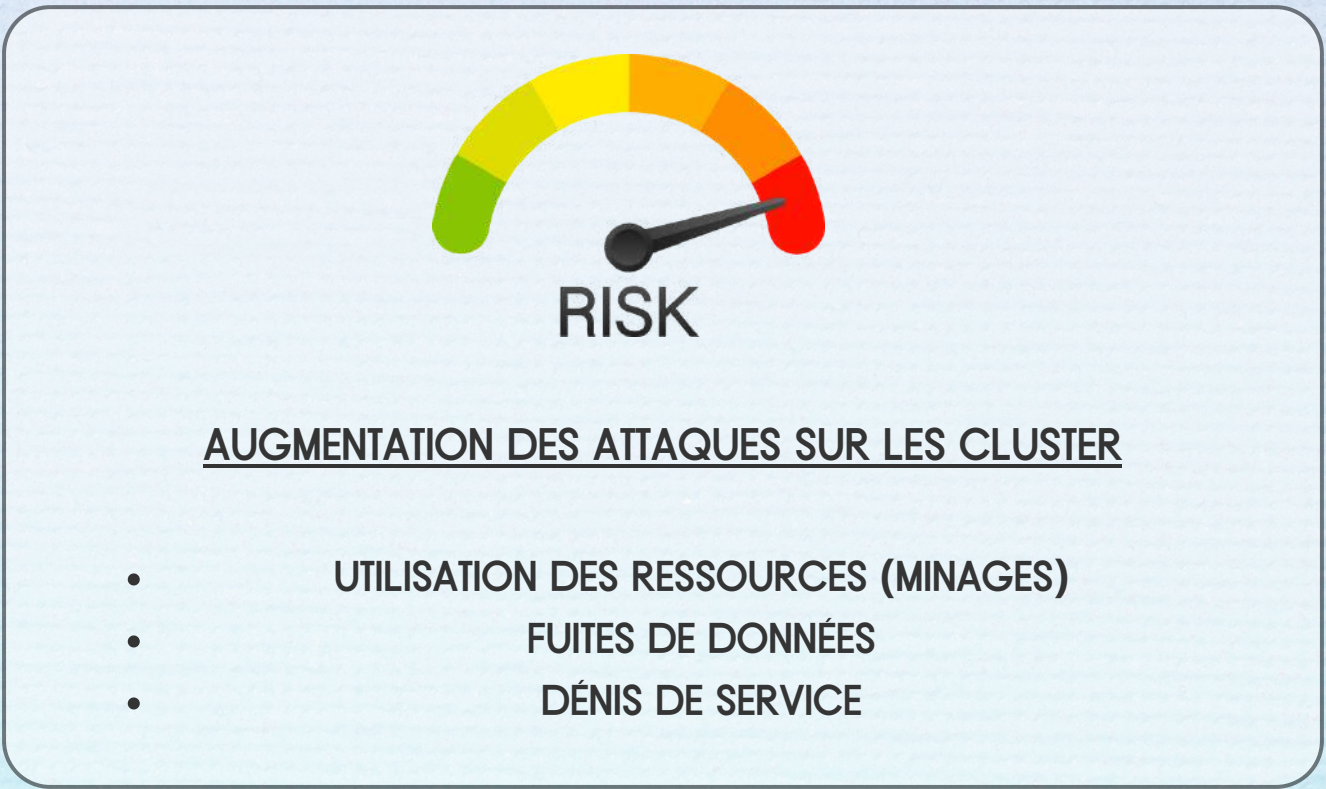
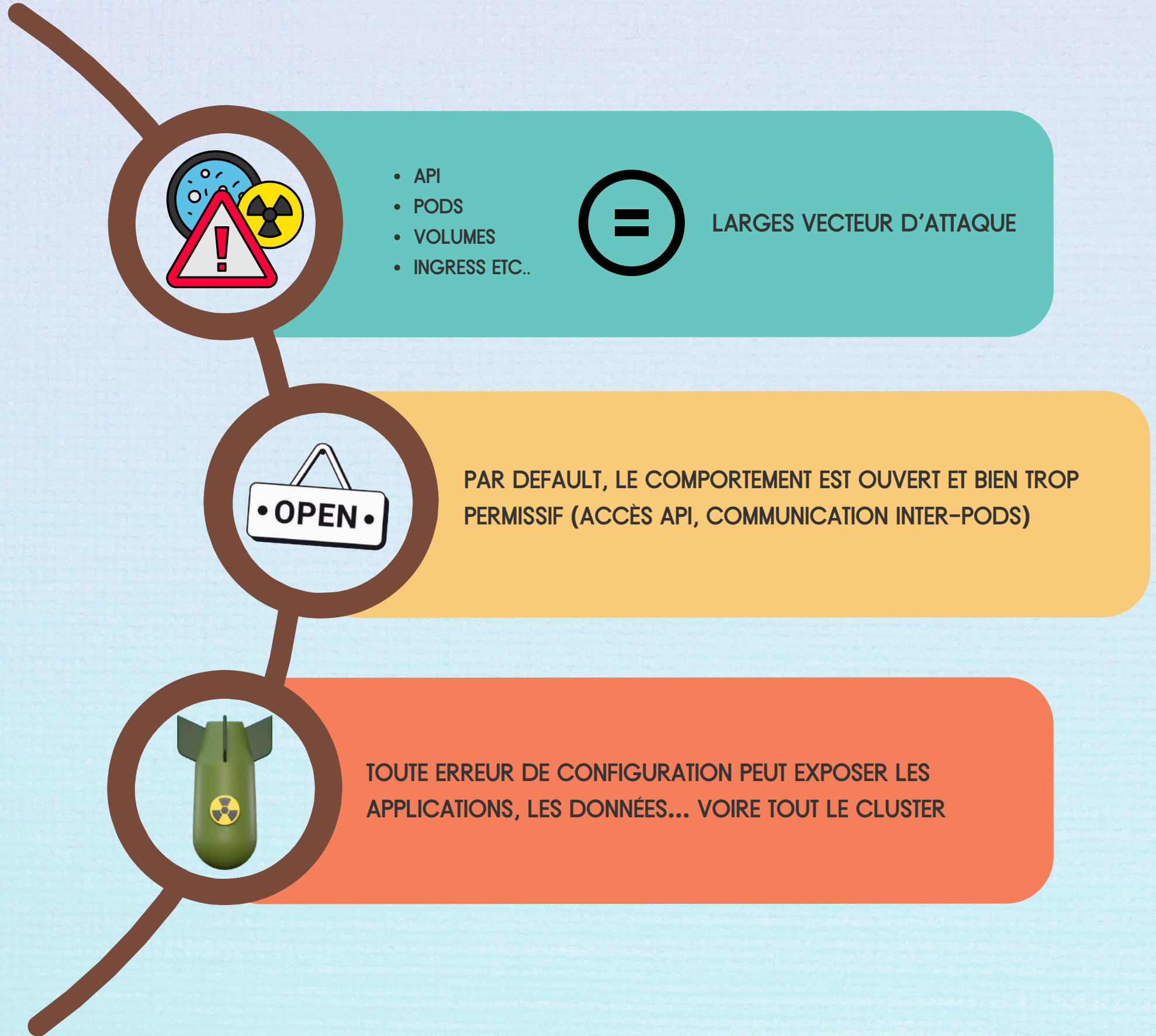
- **Nombreux composants à sécuriser (API, pods, services...)**
- **Par défaut, tout est souvent trop ouvert**
- **Plus la plateforme est puissante, plus la surface d'attaque est grande**

KUBERNETES EN BREF

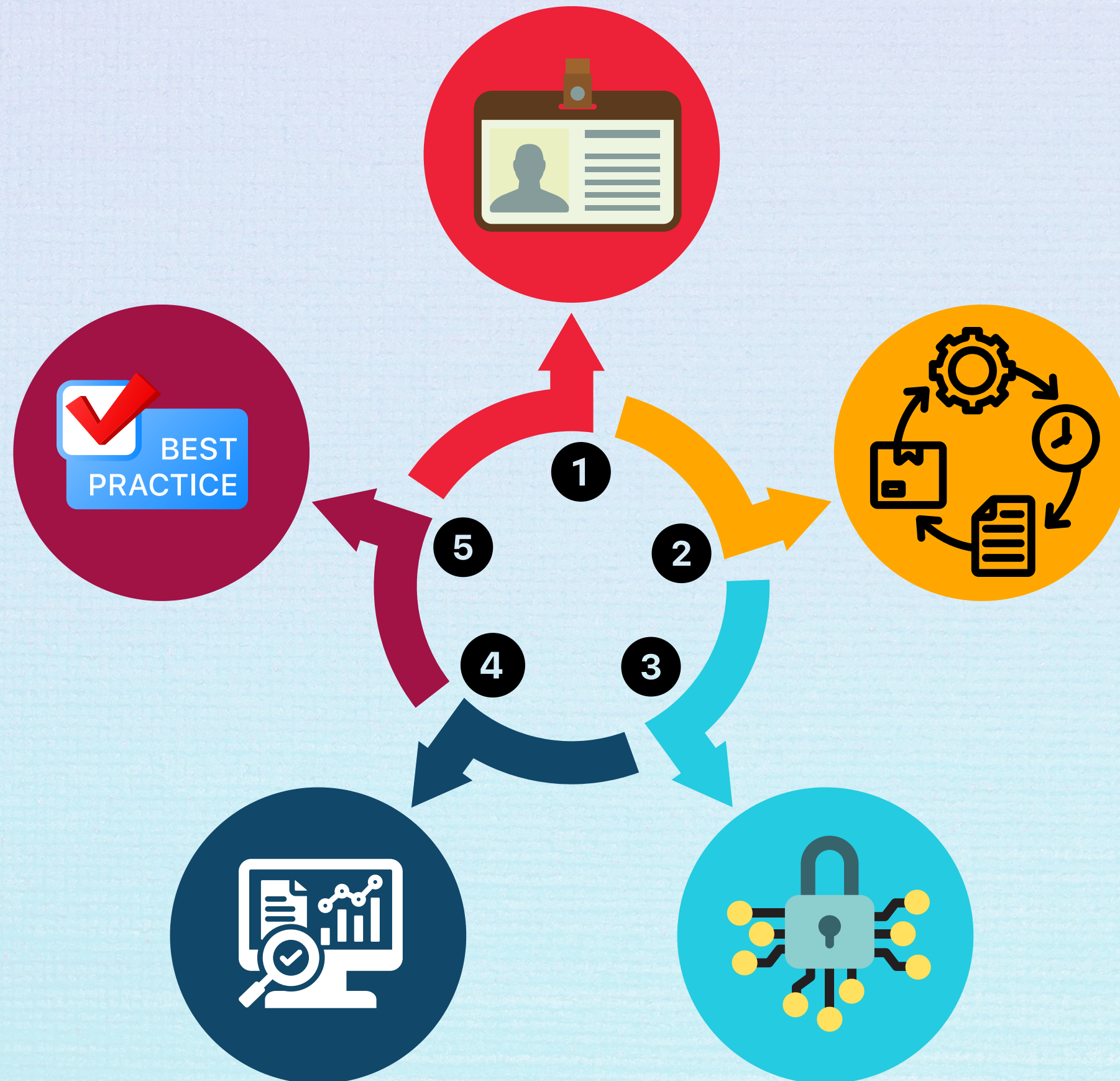
KUBERNETES (K8S), EST UN SYSTÈME OPEN-SOURCE PERMETTANT D’AUTOMATISER LE DÉPLOIEMENT, LA MISE À L’ÉCHELLE ET LA GESTION DES APPLICATIONS CONTENEURISÉES.



POURQUOI LA SÉCURITÉ KUBERNETES EST ESSENTIELLE ?



PRINCIPES FONDAMENTAUX DE SÉCURITÉ KUBERNETES



1) Gestion des identités et des accès

- Authentification dans Kubernetes
- RBAC
- Bonne pratiques

2) Sécurisation des workloads et des conteneurs

- SecurityContext
- Pod Security Standards (PSS)
- Privilèges
- Runtime Security
- Scanning /signatures
- Isolation

3) Protection des communications et des données

- Chiffrement des communications
- Gestion des secrets
- Chiffrement des données

4) Surveillance et détections des menaces

- Audit & logs
- Monitoring & Alerting
- Détection d'intrusions

5) bonnes pratiques et retours d'expériences

- Pratique recommandées
- pièges courants
- Retours d'expériences

GESTION DES IDENTITÉS ET DES ACCÈS

AUTHENTIFICATION DANS KUBERNETES

L’AUTHENTIFICATION

Avant qu’une action soit autorisée dans Kubernetes (ex : créer un pod, lire un secret, modifier une config), le composant central du cluster – l’API Server – doit vérifier l’identité de celui qui fait la demande.

L’utilisateur peut être :



Un utilisateur humain (via kubectl)



un script ou un service (ex: pipeline)



Un composant interne Kubernetes (service account)



Kubernetes n’a pas de base de données interne d’utilisateurs, ni de système d’inscription ou de gestion native des comptes.

Au lieu de ça, il délègue l’authentification à des systèmes externes comme :

- Un provider d’identité (OIDC - Open ID Connect)
- Des Certificats TLS
- Un service externe

La création, désactivation, gestion des utilisateurs (humains ou machines) ne se fait pas dans Kubernetes, mais dans votre IAM d’entreprise, vos fichiers kubeconfig, ou vos systèmes d’auth externes...

Kubernetes ne répond qu’à une seule question :

"Est-ce que cette requête vient d’un utilisateur identifié par ailleurs ?"

Ce comportement est volontaire, pour laisser chaque organisation choisir son système d’authentification, selon ses contraintes (sécurité, SSO, législation...).

GESTION DES IDENTITÉS ET DES ACCÈS

AUTHENTIFICATION DANS KUBERNETES



✓ Méthodes recommandées :

- **OIDC (OpenID Connect) → SSO via Azure AD, Google, GitHub**
- **Certificat TLS personnel → avec kubeconfig sécurisé**

⚠ Bonnes pratiques :

- **Utiliser un provider d'identité externe pour gérer les droits**
- **Protéger le fichier kubeconfig**
- **Rotation régulière des certificats ou jetons**

✗ À éviter :

- **Tokens statiques**
- **Fichiers mots de passe / auth basique**



✓ Méthodes recommandées :

- **ServiceAccount avec token JWT**
- **→ Sécurisé, intégré à Kubernetes**
- **→ Supporte la rotation avec les Projected Tokens**
- **Webhook d'authentification**
- **→ Si on veut déléguer au système IAM de l'entreprise**

⚠ Bonnes pratiques :

- **Ne pas partager les jetons entre services**
- **Limiter les permissions du ServiceAccount (RBAC minimal)**

✗ À éviter :

- **Token statique codé en dur**
- **Secrets dans Git**



✓ Méthodes recommandées :

- **Certificats TLS client**
- **→ Auto-générés et validés par le cluster (via kubelet bootstrap)**
- **→ Rotation automatique supportée**

⚠ Bonnes pratiques :

- **Gérer les certificats avec cert-manager ou via kubeadm**
- **Ne pas exposer ces certificats en dehors du cluster**

✗ À éviter :

- **Auth statique / manuelle non supervisée**

SSO : Single Sign On

TLS : Transport Layer Security

JWT: Json Web Token

GESTION DES IDENTITÉS ET DES ACCÈS

LES RBAC

Le RBAC est un système de gestion des permissions se basant sur des rôles définis, ces rôles déterminent l'accès aux ressources. Le RBAC contrôle l'accès aux API Kubernetes

RBAC (ROLE BASED ACCESS CONTROL)

Les Roles peuvent être attribués à :

- Des utilisateurs
- Des groupes
- Des comptes de service (service account)



Role

Définit les droits dans un namespace



ClusterRole

Définit les droits dans tout le cluster



RoleBinding

Donne un Role dans un namespace



ClusterRoleBinding

Donne un ClusterRole



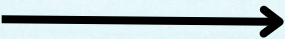
```
kind: ClusterRole
metadata:
  name: lecteur-pods
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list"]
```



```
kind: ClusterRoleBinding
metadata:
  name: binding-lecteur
subjects:
- kind: User
  name: alice
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: lecteur-pods
  apiGroup: rbac.authorization.k8s.io
```



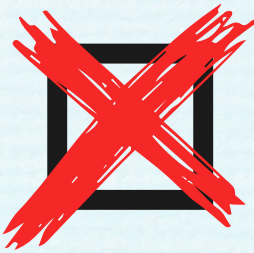
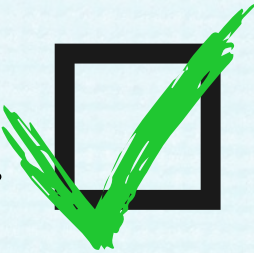
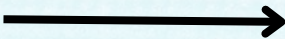
Identité reconnue par Kubernetes



Lecture du Role / ClusterRole



Association Via Binding



Accès autorisé ou refusé

GESTION DES IDENTITÉS ET DES ACCÈS

BONNES PRATIQUES

Appliquer le principe de
moindre privilège.

Donner les droits
nécessaires, rien de
plus.

Créer des rôles
spécifiques par usage
(readonly, ci-cd-runner,
admin-prod..)

Eviter autant que
possible le rôle cluster-
admin.

Séparer les humains des
machines (user,
serviceAccount)

Rotation régulière des
credentials



Audits réguliers



Documentation



OICD 

OICD Externe : Github, Google, AWS Cognito, Azure AD, Auth0...)

OICD interne : Dex, Keycloak

SÉCURISATION DES **WORKLOADS** ET DES CONTENEURS

SECURITYCONTEXT – LA BASE DE LA SÉCURITÉ AU NIVEAU DU POD ET DES CONTENEURS.



Le champ securityContext permet de définir les règles de sécurité appliquées à une image **lorsqu'elle est exécutée dans un pod**. Il contrôle comment le conteneur fonctionne (utilisateur, permissions, capacités...), indépendamment de ce que l'image contient.

```
securityContext:
  runAsNonRoot: true
  readOnlyRootFilesystem: true
  allowPrivilegeEscalation: false
  capabilities:
    drop: ["ALL"]
```

Exemple YAML

- **runAsNonRoot: true**
 - Empêche un conteneur de tourner en root (bon réflexe par défaut)
- **readOnlyRootFilesystem: true**
 - Interdit l'écriture sur le système de fichiers (réduit les risques de persistance)
- **allowPrivilegeEscalation: false**
 - Empêche l'élévation de privilèges via setuid ou autres mécanismes internes
- **capabilities.drop**
 - Supprime des capacités Linux inutiles (ex : NET_RAW, SYS_ADMIN)



- Ne jamais supposer que l'image est "safe" → toujours définir un securityContext
- Toujours commencer par le minimum, et ouvrir seulement si besoin réel
- Vérifier que vos CI/CD ne déploient pas de conteneurs sans restriction

SÉCURISATION DES **WORKLOADS** ET DES CONTENEURS

POD SECURITY STANDARDS (PSS)



Les Pod Security Standards sont un mécanisme intégré à Kubernetes pour empêcher le déploiement de pods non conformes à des règles de sécurité prédéfinies.

LES 3 NIVEAUX DE SÉCURITÉ PROPOSÉS :



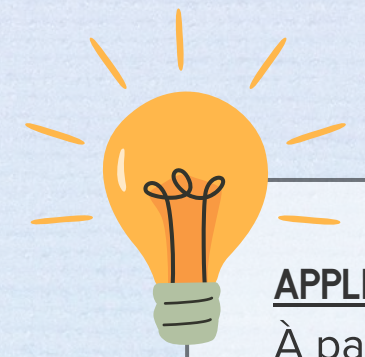
- PRIVILEGED :**
- Aucun contrôle, tout est permis (équivalent à un accès root total)
 - ⚠ Risqué, à éviter



- BASELINE :**
- Recommandé pour la plupart des workloads
 - Interdit les pratiques les plus dangereuses (privileged, hostNetwork...)



- RESTRICTED :**
- Niveau de sécurité maximal
 - Imposent l'usage de runAsNonRoot, seccomp, etc.
 - Adapté aux environnements sensibles / production



APPLICATION :

À partir de Kubernetes v1.25, le mécanisme Pod Security Admission est activé par défaut dans l'API Server.

- Création d'un Namespace
- Application des labels PSS (enforce):

```
kubectl label namespace production \
  pod-security.kubernetes.io/enforce=restricted \
  pod-security.kubernetes.io/enforce-version=latest
```

- Ecrire les YAML de pods conformes aux profils (restricted, baseline..)



- Activer au minimum "baseline" sur tous les namespaces
- Utiliser restricted sur les environnements critiques
- Combiner avec des outils comme Kyverno ou OPA Gatekeeper pour affiner les contrôles

SÉCURISATION DES **WORKLOADS** ET DES CONTENEURS

LES PRIVILÈGES – ÉVITER LES PRIVILÈGES DANGEREUX DANS LES PODS



Objectif : **Réduire la surface d’attaque** en interdisant les comportements risqués par défaut.
Même avec une image propre, un bon securityContext, et un profil Pod Security Standard, certains paramètres Kubernetes permettent encore d’échapper à l’isolation, de monter en privilèges, ou d’interagir avec l’hôte. Il est donc crucial de désactiver explicitement ces options dangereuses et de les surveiller en continu.



PRIVILÈGES À ÉVITER ABSOLUEMENT :

➤ **privileged: true**

Donne un accès root complet à l’hôte (équivalent à un sudo sur le node !). Peut compromettre tout le cluster

➤ **hostPID: true**

Permet de voir les processus du noeud → risque d’espionnage / injection

➤ **hostIPC: true**

Accès à la mémoire partagée de l’hôte → possible fuite ou manipulation

➤ **hostNetwork: true**

Partage le réseau de l’hôte → possible contournement des Network Policies

➤ **allowPrivilegeEscalation: true**

Autorise un conteneur à monter en privilège, même si runAsNonRoot est défini



BEST PRACTICE

- Désactiver tous ces paramètres sauf cas très justifié et documenté
- Toujours passer par une revue de sécurité avant de les autoriser
- Combiner avec un profil PodSecurity en mode restricted pour forcer leur interdiction

```
securityContext:
  privileged: true
  hostNetwork: true
```




```
securityContext:
  privileged: false
  hostNetwork: false
  runAsNonRoot: true
  allowPrivilegeEscalation: false
```



SÉCURISATION DES **WORKLOADS** ET DES CONTENEURS


RUNTIME SECURITY DANS KUBERNETES

 Détecter les comportements anormaux des conteneurs une fois qu'ils sont en cours d'exécution. **Même si un pod a été bien configuré**, et qu'il respecte les securityContext et PSS, **il peut être détourné pendant son exécution**




DANGERS POTENTIELS :


➤ Commandes shell lancées depuis l'intérieur

-  Exemple : un attaquant accède au conteneur avec kubectl exec et lance /bin/bash
➔ Risque : exécution de scripts malveillants, vol de données, persistance


➤ Requêtes réseau inhabituelles

-  Exemple : le pod envoie des paquets vers une IP externe inconnue ou scanne le réseau interne
➔ Risque : exfiltration de données

➤ Accès à des fichiers critiques

-  Exemple : lecture du fichier /etc/shadow, /var/run/docker.sock, /root/.ssh/
➔ Risque : vol de credentials, prise de contrôle du nœud ou du cluster

➤ Tentatives d'évasion de conteneur




-  Exemple : montage de /proc ou usage de capacités SYS_ADMIN, privileged: true
➔ Risque : le conteneur sort de sa “prison” et accède à l'hôte ou au réseau global



BEST PRACTICE

IMPLÉMENTER UN RUNTIME SECURITY

La Runtime Security permet de surveiller les conteneurs actifs, détecter des comportements anormaux, et réagir immédiatement (logs, alertes, blocages).

 OUTIL	 FONCTION PRINCIPALE	 REMARQUES
FALCO	Détection en temps réel via règles (Syscalls)	Maintenu par la CNCF, le plus populaire
TETragon	eBPF natif, surveillance + politique de sécurité	Par Cilium (Isovalent), très puissant
TRACEE	eBPF + sécurité comportementale	Focalisé sur la traçabilité et la sécurité offensive (Aqua)

SÉCURISATION DES WORKLOADS ET DES CONTENEURS

SCANNING DES IMAGES DE CONTENEURS



Objectif : Détecter les vulnérabilités dès la construction de l'image.
Une image de conteneur contient souvent un OS, des bibliothèques, des binaires... C'est autant de composants potentiellement vulnérables
Le scanning d'image permet d'identifier ces failles avant que les images ne soient utilisées en production.



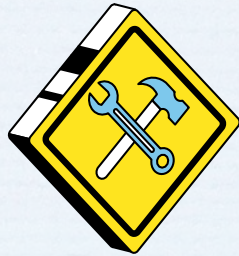
QUE VÉRIFIE UN SCANNER ?

- **Paquets systèmes obsolètes ou vulnérables** (ex : libssl, glibc, etc.)
- **Librairies de langages** (npm, pip, gem...)
- **Mauvaises configurations** (ports ouverts, utilisateurs root, secrets hardcodés...)



BEST PRACTICE

- Intégrer le scan dans le pipeline CI/CD
- Refuser les images avec des failles critiques
- Scanner régulièrement même les images déjà publiées (nouvelles CVE chaque jour)



TOOLS

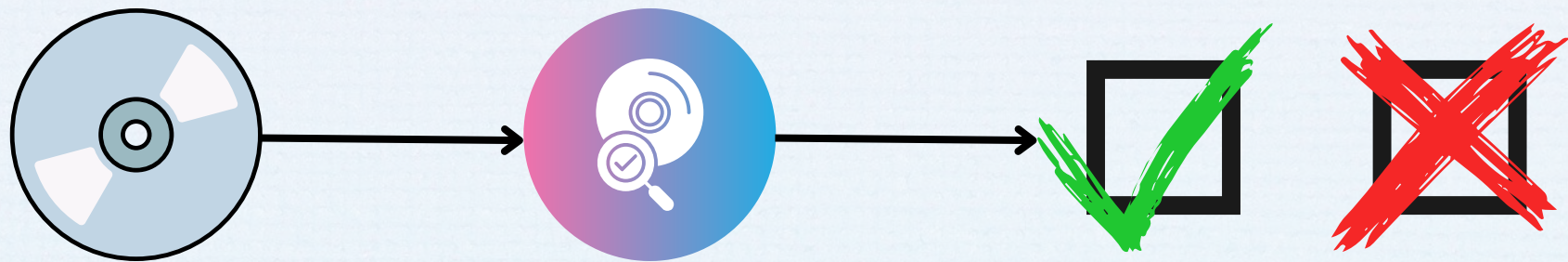
OUTIL	TYPE	POINTS FORTS
TRIVY	CLI / CI	Léger, rapide, supporte Dockerfile, Git repo
GRYPE	CLI / CI	Intégration facile, base Syft
CLAIR	Serveur	Intégration avec registries (Harbor, Quay.io)

SÉCURISATION DES WORKLOADS ET DES CONTENEURS

SIGNATURE DES IMAGES DE CONTENEURS



Objectif : Garantir l'intégrité et la provenance des images déployées
Même si une image est scannée et saine au départ, elle peut être modifiée ou remplacée entre-temps.
La signature cryptographique permet de s'assurer qu'une image n'a pas été altérée et vient bien d'une source de confiance

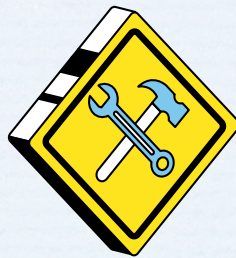


- 1. L'auteur d'une image la signe avec une clé privée
- 2. Le cluster (ou un outil) vérifie la signature avec une clé publique
- 3. Si l'image n'est pas signée (ou mal signée) → elle peut être rejetée





BEST PRACTICE

- Signer toutes les images dans la CI/CD
- Déployer une politique qui rejette les images non signées
- Stocker les clés dans un KMS sécurisé (Vault, GCP KMS, etc.)



TOOLS

 OUTIL	 DESCRIPTION
COSIGN	Projet open-source CNCF, facile à intégrer (Sigstore)
NOTARY V2	Nouvelle version du standard de Docker Content Trust
IMAGEPOLICYWEBHOOK	Permet de bloquer les images non signées (intégré à Kubernetes)

SÉCURISATION DES WORKLOADS ET DES CONTENEURS

ISOLATION RENFORCÉE DES CONTENEURS



Objectif : Limiter au maximum l’impact si un conteneur est compromis.

Par défaut, dans Kubernetes (et Docker), les conteneurs partagent le noyau de l’hôte Linux et sont isolés par des mécanismes comme les namespaces. Cela fonctionne bien en conditions normales mais si un attaquant exploite une faille dans le conteneur (ou dans le noyau), il peut tenter de sortir de son conteneur (évasion) et/ou accéder à l’hôte ou à d’autres conteneurs.



RENFOCER L’ISOLATION

Il ne s’agit pas de remplacer Kubernetes ou Docker, mais d’activer ou d’ajouter des mécanismes complémentaires pour :

- Réduire la liste des actions autorisées (AppArmor, Seccomp)
- Ajouter une “couche intermédiaire” (gVisor, Kata)
- Rendre le conteneur plus autonome, moins dépendant du noyau de l’hôte






SOLUTIONS NATIVES

- **SecurityContext:** `runAsNonRoot`, `readOnlyRootFilesystem`, `allowPrivilegeEscalation: false`
- **Seccomp** → souvent déjà présent, il suffit de l’activer via le securityContext
- **AppArmor** → supporté nativement par beaucoup de distributions (Ubuntu)



SOLUTION EXTERNE / TOOLS

 OUTIL	 TYPE	 USAGE RECOMMANDÉ
GVISOR (GOOGLE)	Runtime sandboxé	Sécurité renforcée sans VM, compatible GKE, facile à intégrer
KATA CONTAINERS	Conteneur dans microVM	Isolation quasi-VM, très utilisé chez Red Hat, Intel, Ant Group
FIRECRACKER	MicroVM ultra-léger	Utilisé par AWS Lambda / Fargate, performant mais plus complexe à déployer soi-même

PROTECTION DES COMMUNICATIONS ET DES DONNÉES

CHIFFREMENT DES COMMUNICATIONS DANS KUBERNETES

Objectif : Protéger les échanges de données entre composants, pods, et services.

Dans un cluster Kubernetes, de nombreuses communications se font entre les utilisateurs (kubectl) et l'API Server, les composants internes (kubelet, scheduler, etc.), les pods entre eux (services web, bases de données...). Toutes ces communications doivent être chiffrées, pour éviter les interceptions et manipulations de données.



CE QUE KUBERNETES CHIFFRE PAR DEFAULT

- Les communications Client → API Server (TLS)
- Kubelet ↔ API Server
- Controller ↔ Scheduler ↔ etcd

Cette couche protège l'infrastructure de contrôle du cluster



CE QUE KUBERNETES NE CHIFFRE PAS PAR DEFAULT

- Les communications entre pods (app ↔ app) ne sont pas chiffrées nativement
- Les pods peuvent parler en HTTP, TCP, UDP... sans sécurité réseau imposée
- Pas d'authentification mutuelle entre pods



mTLS : Version renforcée du protocole TLS où les deux parties (client et serveur) s'authentifient mutuellement à l'aide de certificats.

Service Mesh : Infrastructure logicielle transparente qui gère la communication entre les services d'un cluster (pods), sans modifier les applications.



BEST PRACTICE

POUR SÉCURISER LES ÉCHANGES ENTRE APPLICATIONS



Frontend



Backend



DataBase

- 🔒 Chiffrement bout-à-bout, même entre pods sur le même nœud
- 👤 Authentification mutuelle (mTLS)
- 📋 Traçabilité, audit, contrôle fin des flux

OUTILS É MÉTHODES

- Pour une gestion robuste et automatisée du chiffrement → **Istio** ou **Linkerd**
- Pour un système d'identité standardisé et extensible → **SPIFFE/SPIRE**
- Pour un contrôle simple sans mesh → TLS dans l'app + NetworkPolicies

PROTECTION DES COMMUNICATIONS ET DES DONNÉES

GESTION DES SECRETS DANS KUBERNETES



Objectif : Protéger les données sensibles utilisées par les applications.

Un secret, c'est toute information confidentielle qu'une application utilise (Mot de passe, token API, clé SSH, certificat, etc.)



FONCTIONNEMENT NATIF DES SECRETS DANS KUBERNETES

- Ressource Kubernetes dédiée (kind: Secret)
- Encodés en base64 (⚠ pas chiffrés par défaut sur etcd)
- Montés dans les pods via volumes ou variables d'environnement



LIMITES DU SYSTÈME NATIF

- Base64 n'est pas un chiffrement réel.
- Secrets stockés en clair dans etcd si pas de chiffrement au repos activé
- Risques en cas d'erreur de droits RBAC → fuite potentielle



BEST PRACTICE

- Activer le chiffrement des secrets au repos (EncryptionConfiguration)
- Limiter l'accès aux secrets via RBAC strict
- Ne pas versionner de secrets dans Git
- Éviter de monter des secrets via env: mais préférer le volume (No RAM /Temp)



TOOLS

MÉTHODE	AVANTAGE CLÉ	LIMITE
SECRET + VOLUME	Facile, natif	Pas dynamique, pas audité
SEALEDSECRETS / SOPS	GitOps + chiffrement fort	Pas de rotation automatique
VAULT + CSI	Accès dynamique et temporaire	Complexité de setup
EXTERNAL SECRETS OPERATOR	Synchronisation depuis un vrai KMS (AWS, GCP)	Dépend des clouds



- **Petit cluster / dev local** : Secret monté en volume
- **Sécurité renforcée / prod** : Outils externes comme Vault, SealedSecrets, ESO
- **Intégration GitOps** : SealedSecrets, SOPS
- **Cloud natif** (AWS, GCP) : External Secrets Operator

SURVEILLANCE ET DÉTECTION DES MENACES

POURQUOI SURVEILLER SON CLUSTER KUBERNETES ?

➤ **Objectif : Détecter rapidement les comportements anormaux, incidents ou attaques.**



POURQUOI SURVEILLER SON CLUSTER KUBERNETES ?

Un cluster Kubernetes peut être exposé à de nombreuses menaces :

- 🖥️ Intrusions dans un conteneur
- 🐛 Exploitation de failles logicielles
- 🛠️ Mauvaises configurations
- 🚨 Fuite de secrets, latéralisation dans le réseau
- 🧪 Création de pods ou services suspects
- .

Pour réagir à temps, il faut surveiller l'activité du cluster en continu

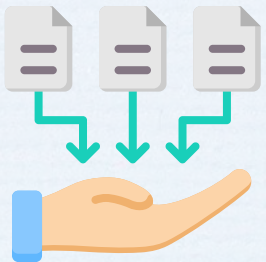


CE QUE L'ON CHERCHE À DÉTECTER

Accès API inhabituel ➤ Fuite de credentials, scans

Création de pods inconnus ➤ Tentative de déploiement malveillant

Connexions réseau anormales ➤ Exfiltration ou mouvements latéraux



Collecter les événements

- Logs, métriques, traces



Analyser le comportement

- Corréler, identifier les anomalies



Détecter les menaces

- Intrusion, exécution anormale, erreurs



Régir

- Alertes, blocage, audit

SURVEILLANCE ET DÉTECTION DES MENACES

AUDIT LOGS DANS KUBERNETES



Objectif : Tracer les actions effectuées sur l'API Kubernetes.

Kubernetes permet de logger chaque requête faite à l'API Server Qui a fait quoi ? Quand ? Depuis quelle IP ? Sur quelle ressource ?
C'est la base pour toute enquête post-incident ou contrôle de conformité.



FONCTIONNEMENT

Kubernetes utilise un fichier de configuration appelé Audit Policy qui :

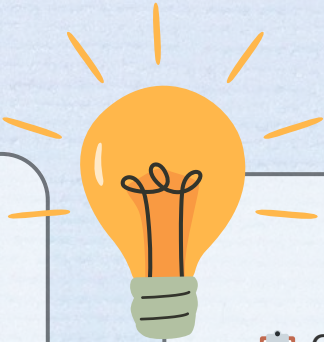
- Définit quelles requêtes doivent être enregistrées
- Précise quel niveau de détail (metadata, request body, etc.)
- Peut filtrer par :
 - utilisateur
 - namespace
 - type d'opération (get, create, delete, etc.)

Cela permet d'éviter de logger "tout" inutilement, tout en ciblant ce qui est critique.

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
- level: Metadata
  verbs: ["create", "delete"]
  resources:
    - group: ""
      resources: ["pods", "secrets"]
```

Ici, on log uniquement les créations et suppressions de pods et secrets.

Par défaut : dans un fichier local sur le master (/var/log/kubernetes/audit.log) mais on peut aussi les envoyer vers une API externe (webhook receiver), les pousser dans un SIEM ou un outil de log (via Fluentd)



POURQUOI C'EST IMPORTANT

- 📋 Conformité (RGPD, ISO) : prouver qui a accédé/modifié une ressource
- 🔍 Forensic : en cas d'attaque ou d'erreur, retracer exactement ce qui s'est passé
- 🔔 Détection : couplé à un outil (ex : SIEM), déclenche des alertes

TOOLS

- **Elasticsearch** : Moteur de recherche et d'analyse distribué, utilisé pour stocker et indexer les logs.
- **Logstash** : collecte, transforme et envoie les logs vers une destination comme Elasticsearch.
- **Kibana** : Interface de visualisation pour Elasticsearch, permettant de créer des tableaux de bord interactifs basés sur les logs collectés.
- **Fluentd** : Collecteur de logs open-source qui unifie la collecte et la consommation de données pour une meilleure observabilité.

SURVEILLANCE ET DÉTECTION DES MENACES

MONITORING DANS KUBERNETES



Objectif : Superviser la santé du cluster, des nœuds, et des applications.

Contrairement aux audit logs, centrés sur les actions API, le monitoring permet de suivre l'état de santé et les performances du cluster et des applications en temps réel.



Mesurer l'état et la performance des composants



Détecter des anomalies (latence, surcharge, crash...)



Avoir une vision temps réel du cluster



CE QUI DOIT ÊTRE SURVEILLÉ

- **Cluster / nœuds** : CPU, RAM, disk, load, kubelet, etcd
- **Pods / containers** : Redémarrages, mémoire, saturation CPU
- **Services applicatifs** : Temps de réponse, erreurs HTTP, latence
- **Jobs / CronJobs** : Échecs, durée d'exécution



TOOLS

Outil	Fonction principale
Prometheus	Collecte des métriques via scrape (pull model)
Grafana	Visualisation de métriques, alerting
Kube-State-Metrics	Expose l'état des objets Kubernetes en métriques
Alertmanager	Gestion des alertes Prometheus



ALERTES RECOMMANDÉES

- Nœud non prêt / saturé
- Pod crashloop ou restart fréquent
- Latence élevée / 5xx HTTP
- Secret expiré ou usage anormal de ressources

SURVEILLANCE ET DÉTECTION DES MENACES

DÉTECTION D'INTRUSION (RUNTIME SECURITY)



Objectif : Identifier les comportements anormaux au moment de l'exécution

Même avec une bonne configuration et du monitoring, certaines attaques peuvent passer inaperçues.

La détection d'intrusion vise à **surveiller les conteneurs en temps réel, détecter des comportements suspects** (exec, scan réseau, escalade de privilèges...) et **générer des alertes ou bloquer les actions**.

EXEMPLES D'ÉVÉNEMENTS DÉTECTÉS :

kubectl exec dans un pod ➤ Intrusion ou debug non autorisé

Accès à /etc/shadow ➤ Tentative de vol de credentials

Lancement de shell (/bin/sh) ➤ Backdoor, exploitation manuelle

Ouverture de socket inhabituel ➤ Lateral movement / exfiltration




**BEST
PRACTICE**

- **Installer un agent runtime léger** (ex : Falco, Tetragon) sur tous les nœuds
- **Filtrer les événements critiques** : exec dans un pod, modification de fichiers système, accès à des sockets réseau
- **Intégrer les alertes** au SIEM ou à Slack/Email
- Limiter les faux positifs en adaptant les règles à ton environnement
- Compléter par des outils de forensic pour investigation post-incident



TOOLS

 OUTIL	FONCTION PRINCIPALE
<u>FALCO</u>	Détection d'événements système en temps réel (CNCf)
<u>TETRAGON</u>	Sécurité comportementale basée sur eBPF
<u>TRACEE</u>	Runtime Security via eBPF (par Aqua Sec)

PRATIQUES RECOMMANDÉES



GOUVERNANCE ET SÉCURITÉ DÈS LA CONCEPTION

- Appliquer les principes du moindre privilège (RBAC minimal)
- Activer Pod Security Standards dès le début du projet
- Gérer les secrets via outils dédiés (Vault, SealedSecrets...)



CONCEPTION DES WORKLOADS

- Toujours définir un securityContext clair
- Restreindre les capacités Linux (capabilities)
- Utiliser readOnlyRootFilesystem + runAsNonRoot



CI/CD ET AUTOMATISATION

- Scanner les images à chaque build (Trivy, Gype...)
- Signer les images (Cosign)
- Valider les manifestes avec des politiques (OPA Gatekeeper, Kyverno)



OBSERVABILITÉ

- Centraliser logs, métriques et alertes
- Mettre en place des alertes sur comportements anormaux
- Utiliser la runtime security (Falco, Tetragon)

PIÈGES COURANTS À ÉVITER

❌ PIÈGE	💣 CONSÉQUENCE POSSIBLE
PODS TOURNANT EN ROOT	Risque d'évasion, accès non contrôlé
SECRETS ENCODÉS BASE64 SANS CHIFFREMENT	Exfiltration simple en cas de fuite etcd
RBAC TROP LARGE (ADMIN SUR TOUT)	Escalade de privilèges, compromission du cluster
HOSTNETWORK: TRUE SANS RAISON	Bypass des NetworkPolicies
PAS DE ROTATION DE TOKEN / CERTIF	Perte de contrôle sur les accès long terme



RETOURS D'EXPÉRIENCE



Article : La sécurité, un obstacle au déploiement de Kubernetes



Article : Meilleures pratiques en matière de sécurité avec Kubernetes



Mauvaises configurations réseau facilitant les mouvements latéraux



Article : Exposition de services internes via des erreurs de configuration



Article : Utilisation d'images de conteneurs vulnérables



Article : Funny : Public failure stories related to Kubernetes



Article : Kyverno



RETOURS D'EXPÉRIENCE



Constats courants en entreprise :

- La sécurité arrive trop tard, une fois les apps déployées
- Beaucoup de secrets traînent dans les repos Git ou CI/CD
- Les équipes ne savent pas “qui a fait quoi, quand”



Ce qui fonctionne bien

- Définir des standards de sécurité dès le début
- Documenter et versionner les rôles et permissions
- Adopter un outil comme [OPA Gatekeeper](#) / [Kyverno](#) pour faire respecter les bonnes pratiques
- Former les développeurs à la sécurité Kubernetes

SUPERVISION & MONITORING

PROMETHEUS & GRAFANA



```
serviceMonitorSelector: {}
serviceMonitorNamespaceSelector: {}
podMonitorSelector: {}
podMonitorNamespaceSelector: {}
```

RÈGLES DE SCRAPING

```
defaultRules:
  create: true
  rules:
    k8s: true
    node: true
    kubelet: true
    kubernetesSystem: true
```

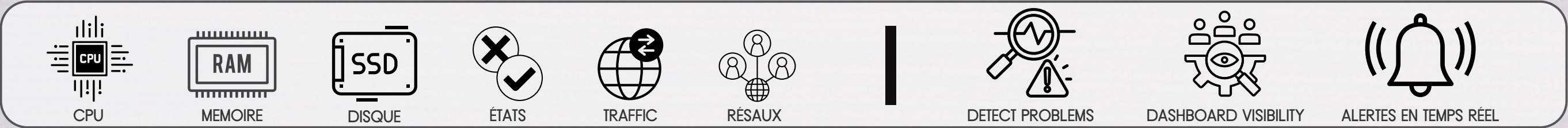
RÈGLES D'ALERTE PRÉDÉFINIS (COMPOSANTS, CPU, MÉMOIRE, DISQUE, ÉTATS, SYSTEM) - EMAIL OU SLACKS

```
nodeExporter:
  enabled: true
```

NODES EKS : CPU, MÉMOIRE, DISQUE, TRAFFIC, CHARGE, T° ..)

```
monitoring:
  enabled: true
  metrics:
    - mongodb_connections
    - mongodb_memory
    - mongodb_storage
  serviceMonitor:
    enabled: true
    namespace: production
```

MONITORING BDD : CONNEXIONS, OPÉRATIONS, STOCKAGE (30SEC)



SÉCURISATION DES DÉPLOIEMENTS

LES DIFFÉRENTES COUCHES DE SÉCURITÉ.

SÉCURITÉ DES CONTAINERS

