# Sécurité des infrastructures basées sur Kubernetes

Xavier Mehrenberger

2020-06-05

AIRBUS

# Introduction

# Introduction

## Pourquoi parler de Kubernetes (`k8s`) ?

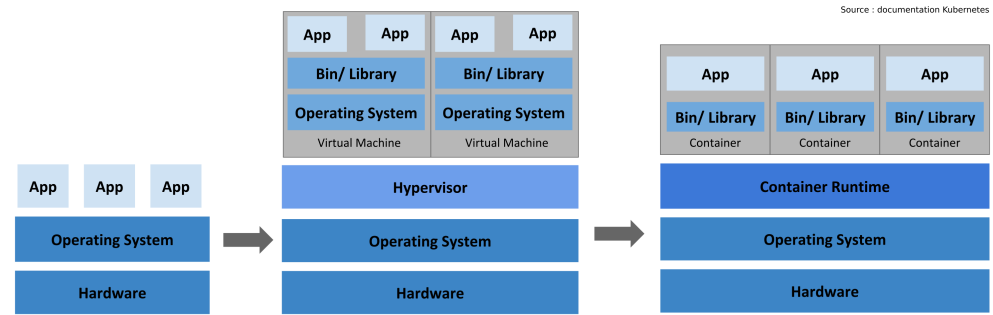
- Orchestrateur de conteneurs populaire (limites conseillées : 5k machines, 300k conteneurs)
- Apporte des fonctionnalités de sécurité intéressantes
- Comment évaluer la sécurité d'un cluster Kubernetes ?
- Comment concevoir, construire un cluster sécurisé ?

## Plan

- Présentation de Kubernetes
  - Ressources et API
  - Machinerie de Kubernetes
  - Exemple d'application volontairement non sécurisée
- Sécurisation d'un cluster
- Sécurisation d'applications hébergées

# Présentation de Kubernetes

# Infrastructure utilisant des conteneurs






AIRBUS

# Quelques avantages de Kubernetes

## Fonctionnalités

- Exécution d'une application sur plusieurs `Nodes`
- Assurer l'adéquation entre nombre d'instances d'une application et charge
- Relancer les containers en cas de crash
- Faciliter les montées de version progressives
- Faciliter le *rollback*

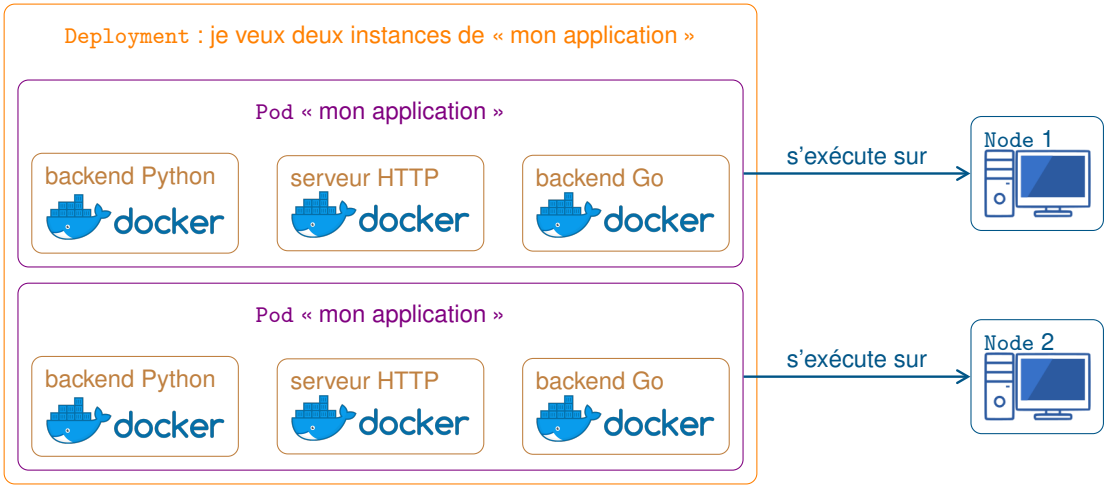
## *Infrastructure as Code*

- « Recettes » de construction de conteneurs
- Approche déclarative pour les ressources Kubernetes
- Infrastructure immutable : plus reproductible

## Dockerfile

```
FROM debian:buster
WORKDIR /install
RUN apt-get update
RUN apt-get install python -y
CMD python3 http.server
...
```

# Exemple de déploiement






AIRBUS

# Objets de l'API

## Objets de base

- `Namespace` : la plupart des objets appartiennent à un espace de noms
- `Pod` : ensemble de conteneurs
- `Service` : permet de rendre l'application accessible par le réseau

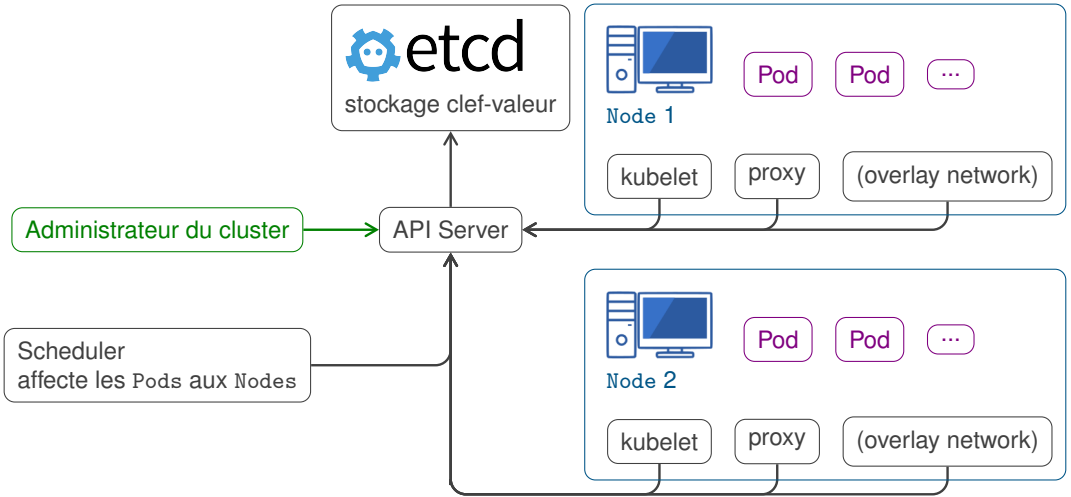
## Objets de plus haut niveau : exécution de `Pods`

- `Deployment` : exécution d'une ou plusieurs instances d'un `Pod`
- `DaemonSet` : `Pods` exécutés sur chaque `Node`
- `Job`, `CronJob` : tâches à la demande, périodiques

## Objets de configuration

- `ConfigMap` : stockage de tables de configuration
- `Secrets` : stockage de secrets de petite taille
- `Role`, `RoleBinding`, `ClusterRole`, `ClusterRoleBinding` : déclaration de permissions
- `NetworkPolicies` : déclaration de règles de filtrage réseau

# Machinerie de Kubernetes – cinématique de déploiement



# Exemple d'application non sécurisée – Dockerfile

## Dockerfile

```
# Cette nouvelle image est construite à partir de l'image publique debian:buster
FROM debian:buster
# Créer le répertoire /install et travailler dedans
WORKDIR /install
# Installer le package wget
RUN apt-get update && apt-get install wget -y
# Télécharger et extraire le binaire du webshell gotty
RUN wget 'https://github.com/yudai/gotty/releases/download/'\
'v1.0.1/gotty_linux_amd64.tar.gz' && \
   tar -xf gotty_linux_amd64.tar.gz
# Définir la commande à executer
CMD /install/gotty --permit-write /bin/bash
```

## Construction de l'image

```
$ docker build -t gotty:demo-sstic gotty
```

# Exemple d'application non sécurisée – objet `Deployment`

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sstic-gotty
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: sstic-gotty
  template:
    metadata:
      labels:
        k8s-app: "sstic-gotty"
    spec:
      containers:
        - name: sstic-gotty
          image: "gotty:demo-sstic"
          securityContext:
            privileged: true
```



AIRBUS

# Exemple d'application non sécurisée – objet `Service`

```yaml
apiVersion: v1
kind: Service
metadata:
  name: sstic-gotty
spec:
  selector:
    k8s-app: sstic-gotty
  ports:
    - name: sstic-gotty-http
      protocol: TCP
      port: 8080
      targetPort: 8080
  type: LoadBalancer
```

# Sécurisation d’un cluster

# Périmètres de responsabilité

## Modes de déploiement

- *Kubernetes-as-a-service* : applications et données seulement
  - ex. Amazon Elastic Kubernetes Service, Google Kubernetes Engine
- *Infrastructure-as-a-service* : idem + installation `k8s`, système d'exploitation
  - ex. location de machines virtuelles + installation avec Kubespray
- Sur site : idem + machines physiques, réseau

## Machinerie `k8s` additionnelle

- *Overlay network* : communications réseau entre `Nodes`
- Stockage
- *LoadBalancer* : se charge d'acheminer le trafic vers un `Service`

# Durcissement du système d'exploitation

## Inventaire des composants logiciels installés sur les machines

- Veille et mise à jour
- Permissions accordées à chaque composant

## Identification des administrateurs

- Qui peut prendre le contrôle de la machine ?
  - Opérateur cloud
  - Administrateurs de l'infrastructure de virtualisation
  - Comptes locaux
  - Etc.

## Hygiène de base

- Pare-feu pour les services locaux
- Appliquer les guides de sécurisation (ex. RedHat, ANSSI)

# Gestion des secrets

## Recenser

- Authentification (ex. TLS entre composants de `k8s`)
- Utilisés par les applications

## Protéger

- Objets de type `Secret`, mis à disposition des conteneurs qui en ont besoin
- Vérifier qu'ils sont utilisés
- Configurer le chiffrement au repos (*at rest*) de `etcd`
  - S'assurer que les secrets servant au chiffrement de `etcd` sont bien protégés
- Application de gestion de secrets (ex. HashiCorp Vault)
  - S'intéresser aux mécanismes et secrets d'authentification

# Configuration de `API Server`

- Certaines permissions par défaut sont trop importantes (peu de conséquences)

## Authentification anonyme

- Par défaut, connexions anonymes autorisées
- ↪ Flag `--anonymous-auth=false` pour désactiver

## Permissions de `kubelet`

- Par défaut, peut modifier des objets concernant un autre `Node` (machines)
- ↪ Fonctionnalité `NodeRestriction` pour désactiver

# Activer la protection `seccomp`

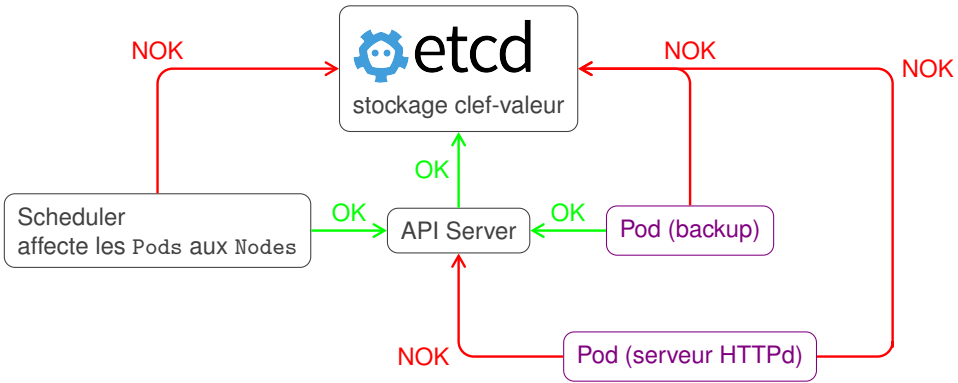
- `Seccomp` : liste blanche d'appels systèmes autorisés
- Activée par défaut avec Docker...
- ...mais pas avec `k8s`
- Peut être activé par `Pod`, par conteneur, ou à l'échelle du cluster

## Exemple

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    # utiliser le profil "hardened.json" pour tous les conteneurs du Pod
    seccomp.security.alpha.kubernetes.io/pod: >
        "localhost/hardened.json"
    # utiliser le profil seccomp par défaut fourni
    # par docker pour le conteneur nommé "conteneur1"
    container.security.alpha.kubernetes.io/conteneur1: >
        "runtime/default"
...
```

# Filtrage réseau

- Déclarer des objets `NetworkPolicies`
  - Accès à `etcd` (`API Server` seulement)
  - Accès à `API Server` (pour applications qui en ont besoin seulement)



# Politique de sécurité à l'échelle du cluster

- Particulièrement utile si le cluster n'est pas utilisé seulement par ses administrateurs
- Sinon, c'est un bon filet de sécurité

## Mécanisme `PodSecurityPolicy`

- Interdire l'accès aux *namespaces* Linux de l'hôte
- Interdire l'utilisation du compte `root` dans les conteneurs
- Choisir le profil `seccomp` appliqué par défaut

Démo : lecture de fichiers dans l'image d'un autre conteneur

# Revue des autorisations

## Fonctionnalités

- 2 types de comptes
  - Compte de service
  - Comptes utilisateurs
- `Role`, `ClusterRole`
  - Liste d'actions autorisées
- `RoleBinding`, `ClusterRoleBinding`
  - Lie un `Role` à compte

## Revue

- Lister tous les `Roles` et `RoleBindings`
- Regarder comment un attaquant pourrait s'en servir

# Sécurisation des applications hébergées

# Filtrage réseau – fonctionnalités

## Objets `NetworkPolicy`

- Autorise le trafic entre deux `Pods`, ou un `Pod` et un range d'IPs
- `Pods` spécifiés par leurs *labels* (métadonnée)
- Protocole (TCP, UDP), port
- Sens entrant (*ingress*) ou sortant (*egress*)

## Application des règles

- Machinerie supplémentaire
  - Overlay network (ex. cilium, calico)
  - Pris en charge par le fournisseur cloud

# Filtrage réseau – exemple

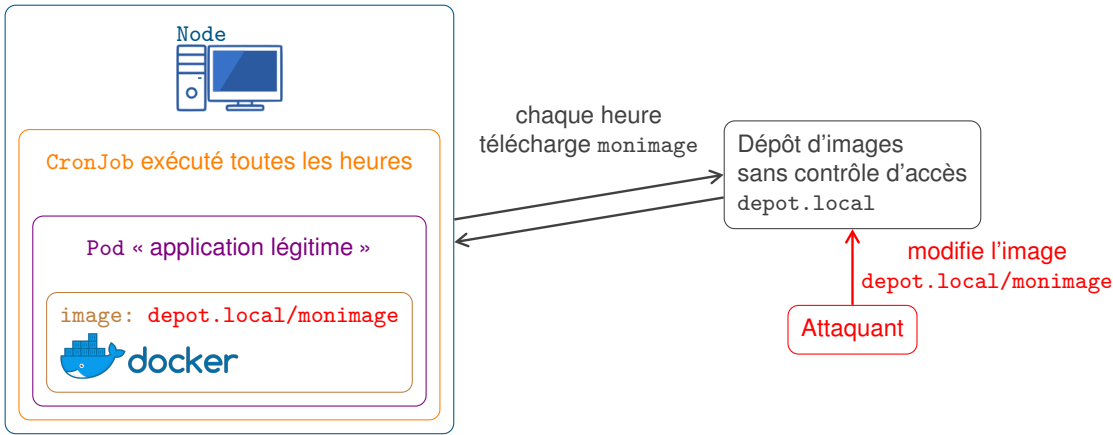
```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: api-allow-mysql-ingress
spec:
  podSelector:         # Cette politique s'applique à tous
    matchLabels:       # les Pods ayant le label
      k8s-app: mysql   # k8s-app: mysql (ex. serveur MySQL)
  ingress:  # cette politique ne concerne
  - ports:  # que les flux entrants
    - port: 3306
      protocol: TCP  # optionnel
    from:                    # le trafic provenant
    - podSelector:           # des Pods ayant le label
        matchLabels:         # uses-mysql: true sera
          uses-mysql: true   # autorisé par cette politique
```

# Comptes de service

- Par défaut, le jeton du compte de service `default` est accessible dans chaque conteneur
- Dispose de peu de permissions par défaut
- ↪ Paramètre `automountServiceAccountToken: false`

Démo : compte de service par défaut accessible

# Gestion des images Docker – attaque possible



# Gestion des images Docker – bonnes pratiques

## Construction d'images

- Utiliser des images de base de confiance
- Vérifier le contenu des images après construction
- Signer les images

## Acheminement et utilisation

- Configurer TLS
- Permissions de lecture/écriture sur le dépôt d'images
- Configurer le cluster pour n'utiliser que des images signées

# Gestion des logs

## Collecte des logs des conteneurs – recettes

- Sorties `stderr`, `stdout`
- Fichier local : adjoindre un conteneur dit *sidecar* dans chaque `Pod`, partager un répertoire
- `syslog` dans `/dev/log` seulement : il faut ruser (cf. actes)

## Centralisation des logs

- Utiliser un système tiers
- Ex. `DaemonSet` collectant les fichiers dans `/var/log/containers/`

# Quotas et limitations de ressources

## Fonctionnalités (API)

- Limiter le nombre de cœurs CPU utilisés
- Limiter la quantité de RAM utilisable

## Implémentation

- Sous Linux, le mécanisme kernel de `cgroups` est utilisé
  - Concrètement, `k8s` demande au Container Runtime de respecter ces limites
    - Le Container Runtime configure les `cgroups`

# Affectation `Pods`/`Nodes`

## Fonctionnalités

- Exprimer des contraintes, respectées par le *scheduler*
- Affinité, anti-affinité

## Scénarios d'utilisation

- S'assurer que deux applications soient sur le même `Node` pour des raisons de performance
- Séparer la machinerie Kubernetes des applications
- Réserver un `Node` à un client (ex. évitera les attaques par canaux cachés CPU)

# Conclusion

# Conclusion – synthèse des fonctionnalités de sécurité

## Fonctionnalités de sécurité intéressantes

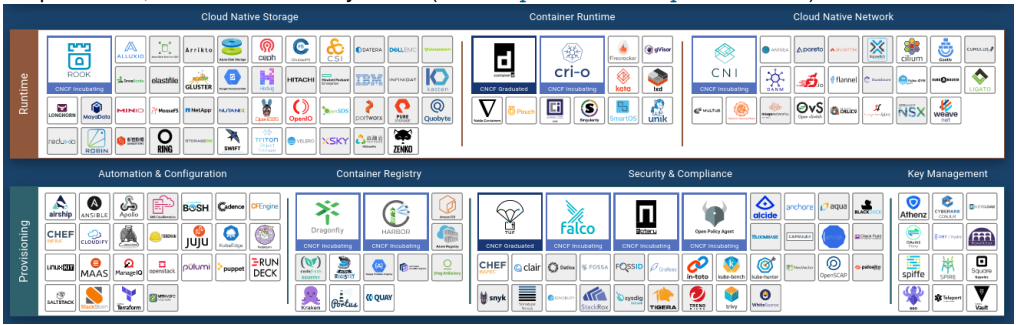
- Infrastructure immutable
- Règles `seccomp`
- Filtrage réseau

## Outils d'analyse automatique

- Images de conteneurs
- Sécurité d'un cluster disponible (ex. kubesec, kube-bench)

# Conclusion – diversité, flexibilité… et analyse de sécurité

- Cas présentés ici assez simples
- Utilisation de Kubernetes : diversité importante à ne pas sous-estimer
- En particulier, diversité de l'écosystème (voir `https://landscape.cncf.io/`)




- Permet d'adapter un cluster à ses besoins
  - Ex. remplacer Docker par gVisor ou Kata
- ↪ Il faut maîtriser tous les composants