# 8. Infrastructure as Code

# 8. Infrastructure as Code

## Intro Infra as Code

**Principe :** gérer des ressources IT (config OS, VM, firewall, stockage, etc.) par l'intermédiaire de fichiers de déclaratifs de descriptions.

**Autrement dit :** j'écris la cible à atteindre, puis la solution d'infra as code se débrouille pour l'atteindre.

# 8. Infrastructure as Code

## Infra as Code *VS.* code impératif

***"Oui, mais on a déjà {bash, Python, …} pour faire cela".***

- Alors, pas exactement.

En langage impératif, on indique quelle action réaliser.

Par ex. installer nginx : `apt install nginx`

A la différence du langage déclaratif, où on indique la cible à atteindre :

```
package: name=nginx state=present
```

# 8. Infrastructure as Code

## Infra as Code *VS.* code impératif

L'intérêt du déclaratif par rapport à l'impératif :

1. D'abord, l'état actuel de la cible est testé.
2. Suivant le mode (juste tester ou appliquer), l'éventuel écart entre l'état déclaré et l'état actuel est corrigé.
   (Autrement dit, le package est installé s'il n'était pas déjà présent, ou bien il ne se passe rien.)
3. Enfin, un rapport sur l'état initial, l'état final, l'éventuelle modification et/ou l'échec d'une modification est remontée à la solution d'Infra as code.

# 8. Infrastructure as Code

## Plusieurs catégories d'Infra as code

1. **Gestionnaire de configuration, provisionning de configuration.**

   Autrement dit du "config as code" utilisé pour configurer l'OS, les services (web, bdd, interpréteur) voire déployer du code.
   → Agit à l'intérieur de la VM.

   Exemples de solutions : Ansible, Puppet, Chef.

# 8. Infrastructure as Code

## Plusieurs catégories d'Infra as code

2. **Gestionnaire d'infrastructure virtuelle.**

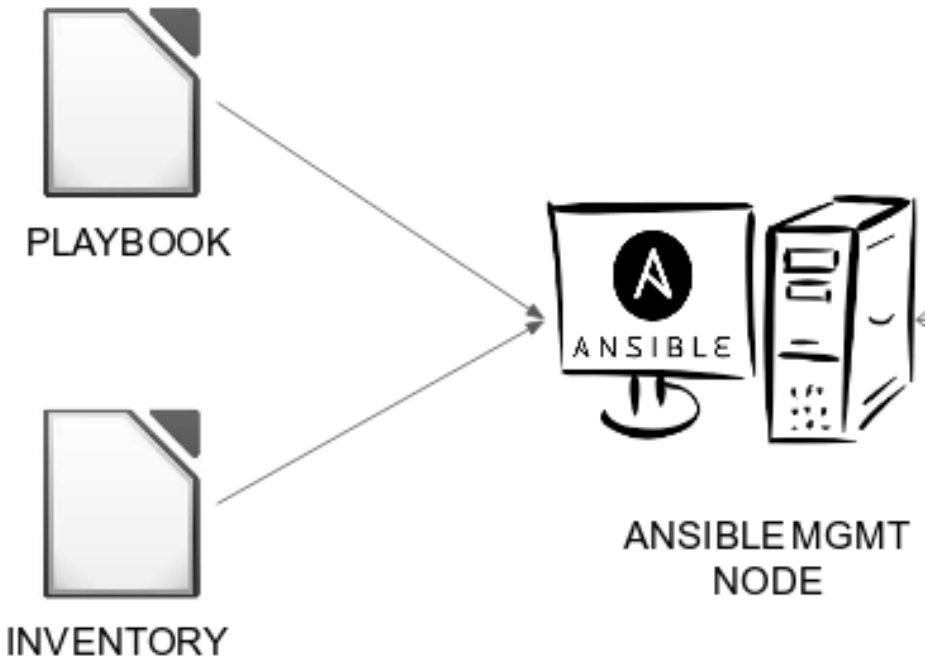
   Instancie les VM, les firewalls (Security Group), la topologie réseau virtuelle, etc.

   → Agit donc à l'extérieur de la VM et bien au-delà.

   Exemples de solutions : Terraform, CloudFormation (AWS).

# 8. Infrastructure as Code

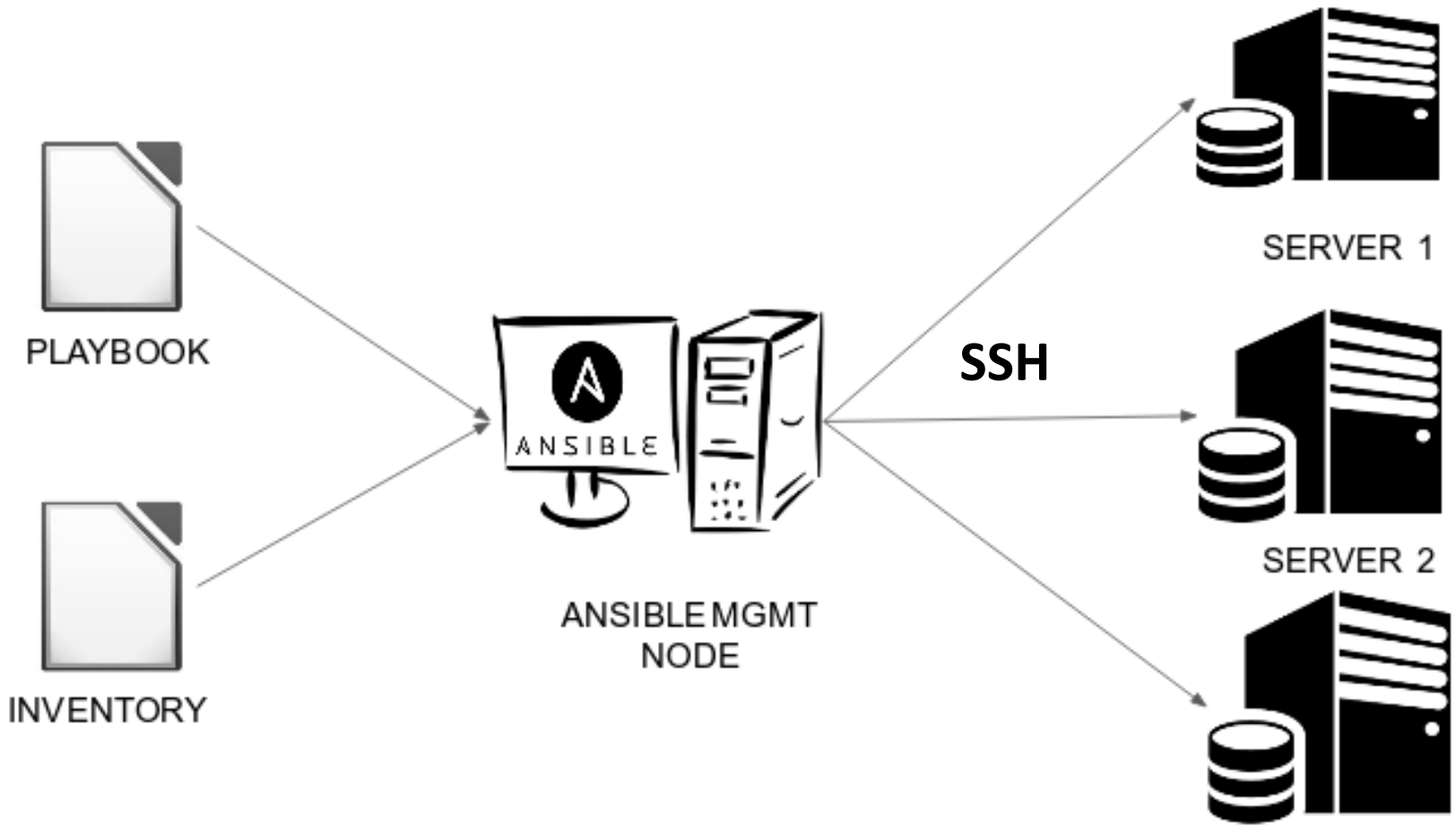
## Ansible




https://medium.com/formcept/configuration-management-and-continuous-deployment-cd0892dce998

# 8. Infrastructure as Code

## Ansible




https://medium.com/formcept/configuration-management-and-continuous-deployment-cd0892dce998

# 8. Infrastructure as Code

## Avantage de l'Infra as Code

**Gain pour le fonctionnel :**

- Automatisation des déploiements → rapidité, productivité, scaling
- Application exacte de la cible déclarer (pas "d'erreur de frappe")
- Meilleure capitalisation de l'expérience pour améliorer les futurs playbooks
- Rollback (retour arrière) plus facile et rapide
- Capacité à tester à l'identique sur une infra de test avant de pousser en prod.

**Gain pour la sécurité :**

- Déjà tous les points ci-dessous.

# 8. Infrastructure as Code

## Avantage de l'Infra as Code

**Gains additionnels pour la sécurité :**

1. Capacité à automatiser les mises à jour logicielles.

2. Capacité à automatiser le renforcement de configuration :
   Plutôt que d'écrire dans Word comment renforcer une VM, écrivons des playbooks de renforcement automatisé.

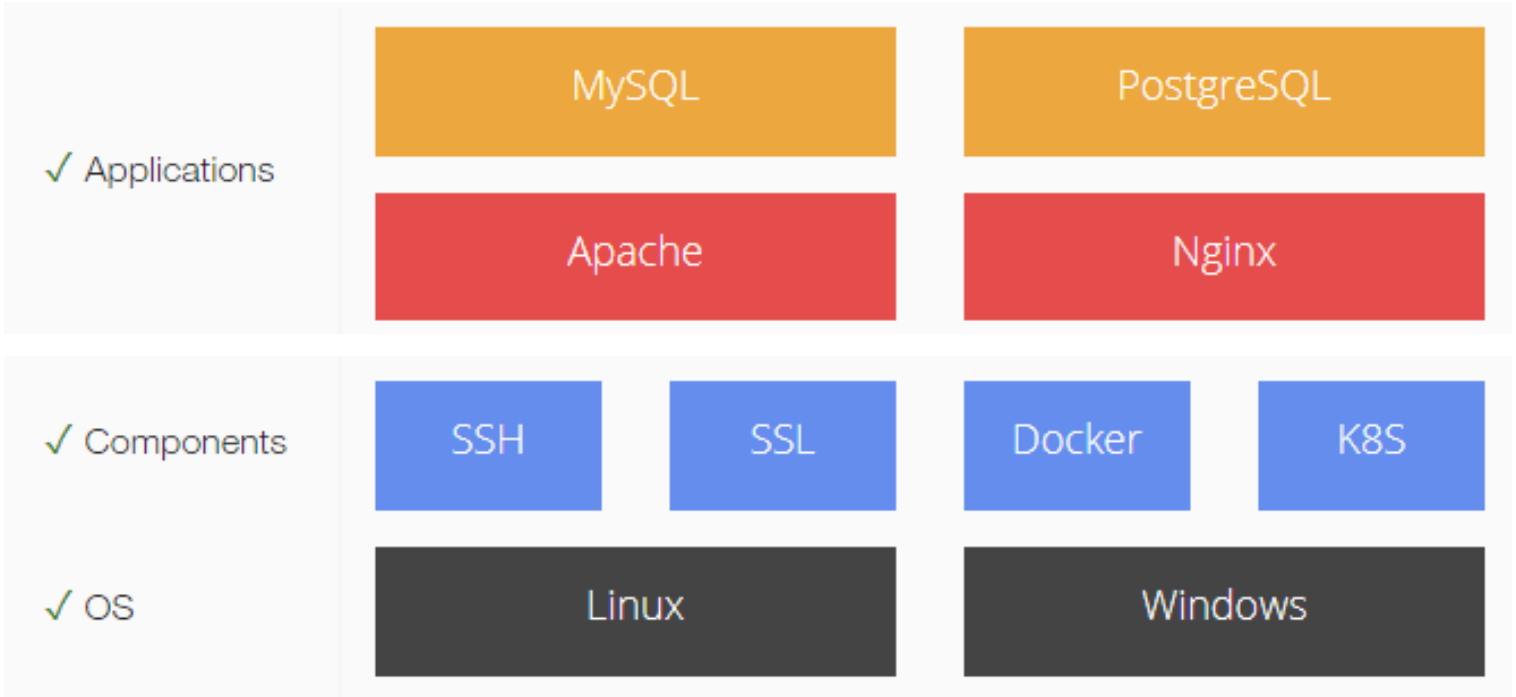
3. Possibilité pour la sécu d'aller faire une revue, voire proposer des modifications, dans les playbooks d'automatisation.

# 8. Infrastructure as Code

## Automatisation du renforcement

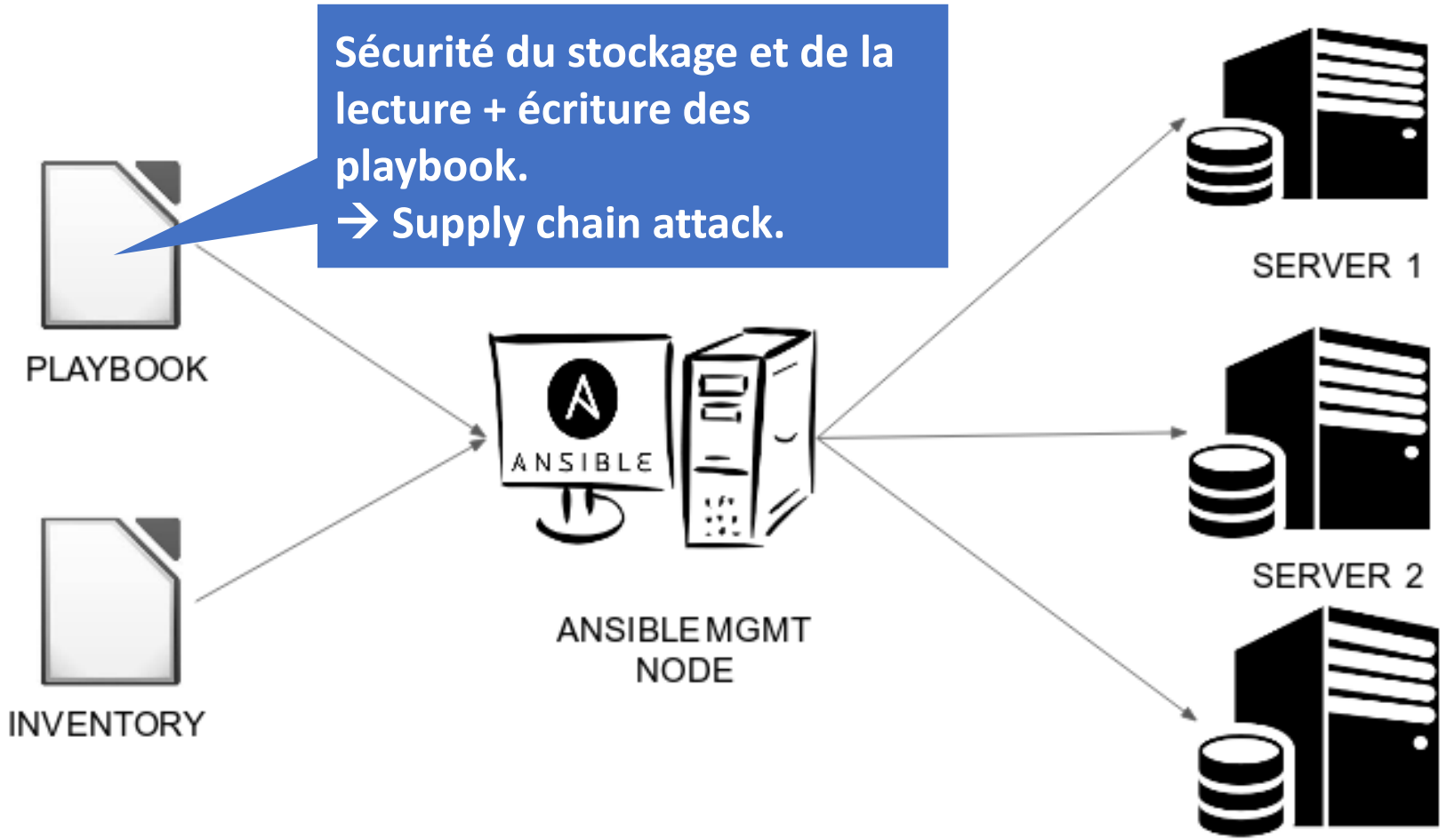
DevSec Hardening Framework

https://dev-sec.io



# 8. Infrastructure as Code

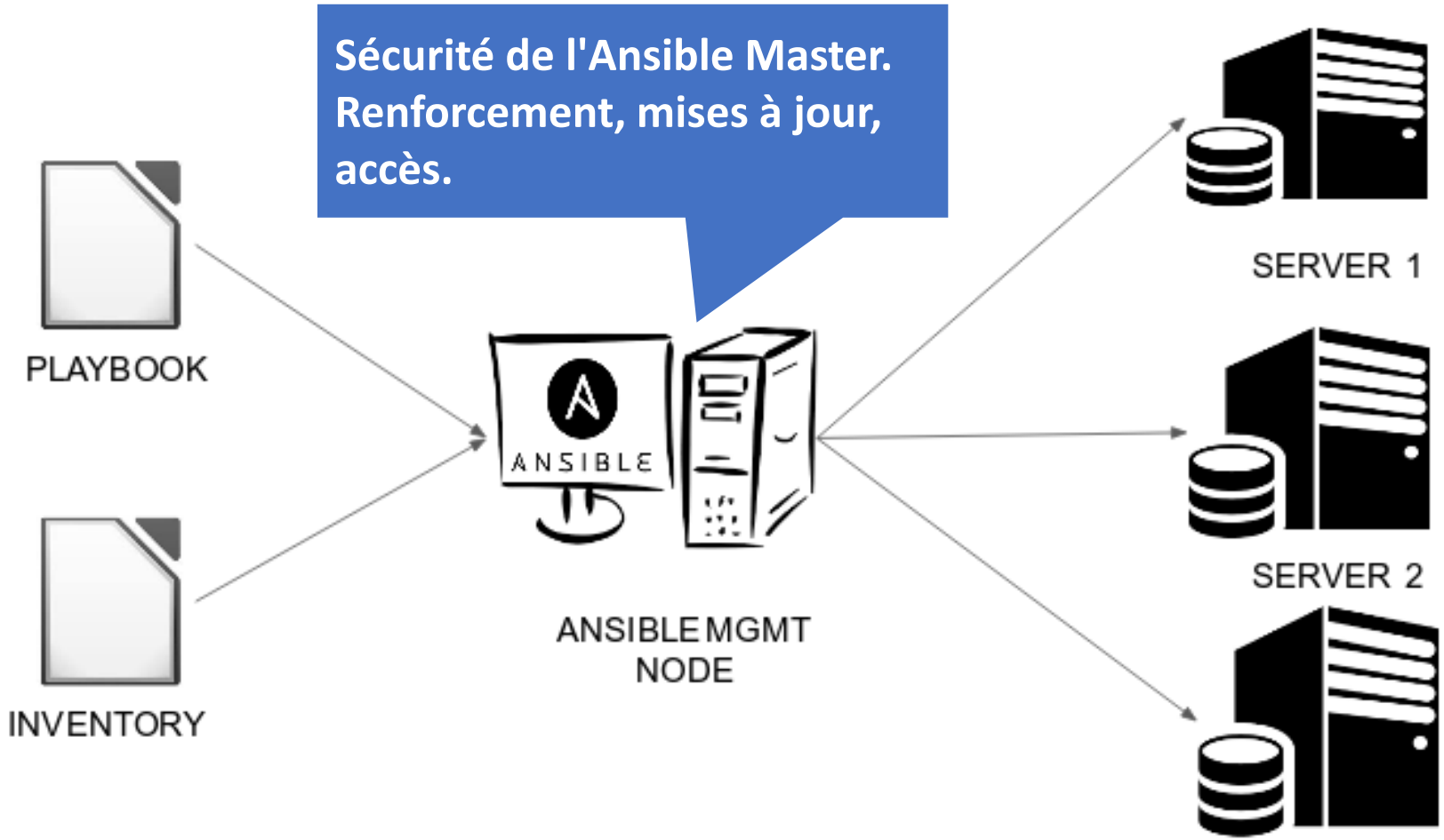
## Risques et vigilances par vis-à-vis de l'infra as code




https://medium.com/formcept/configuration-management-and-continuous-deployment-cd0892dce998

# 8. Infrastructure as Code

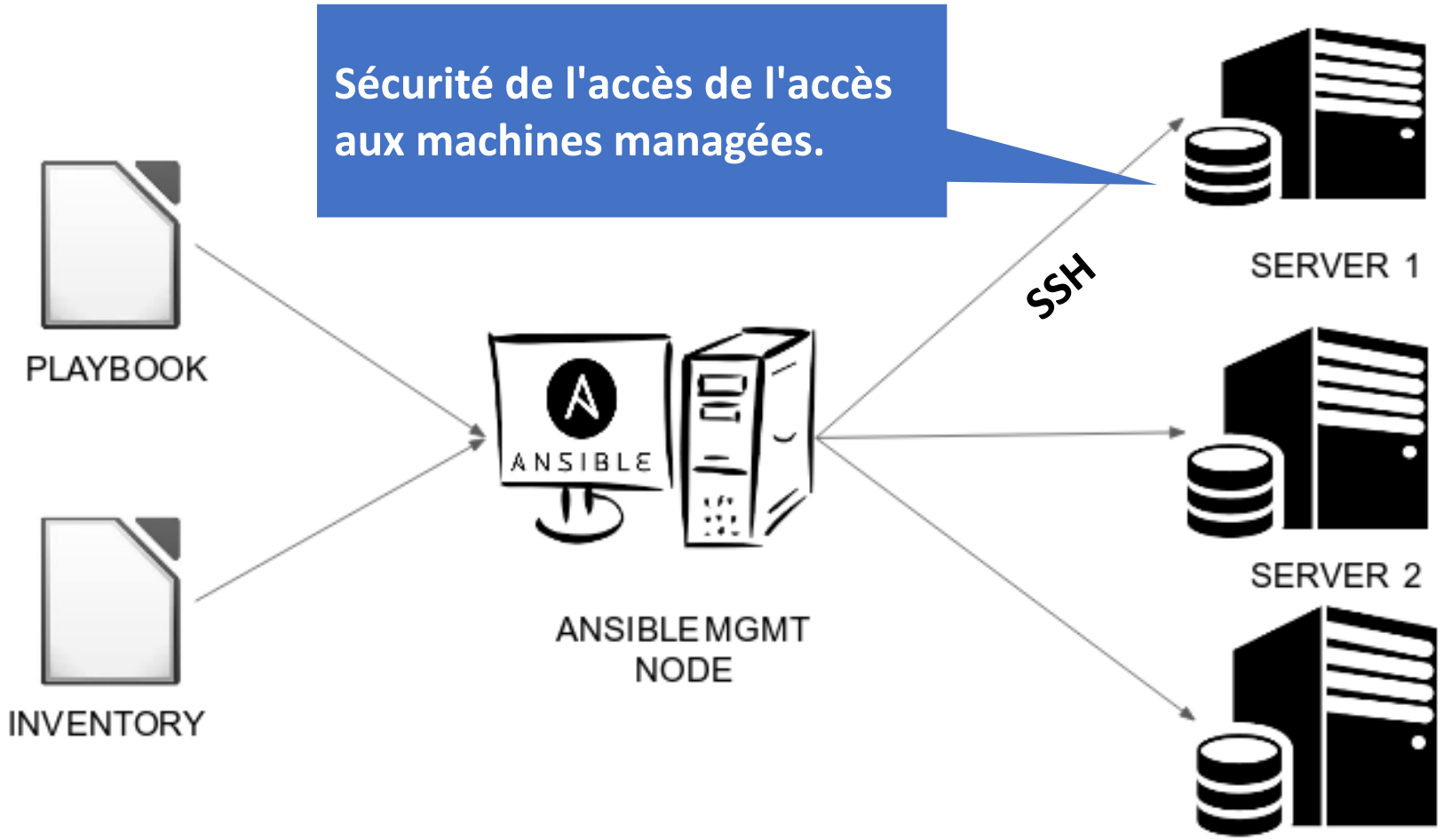
## Risques et vigilances par vis-à-vis de l'infra as code




https://medium.com/formcept/configuration-management-and-continuous-deployment-cd0892dce998

# 8. Infrastructure as Code

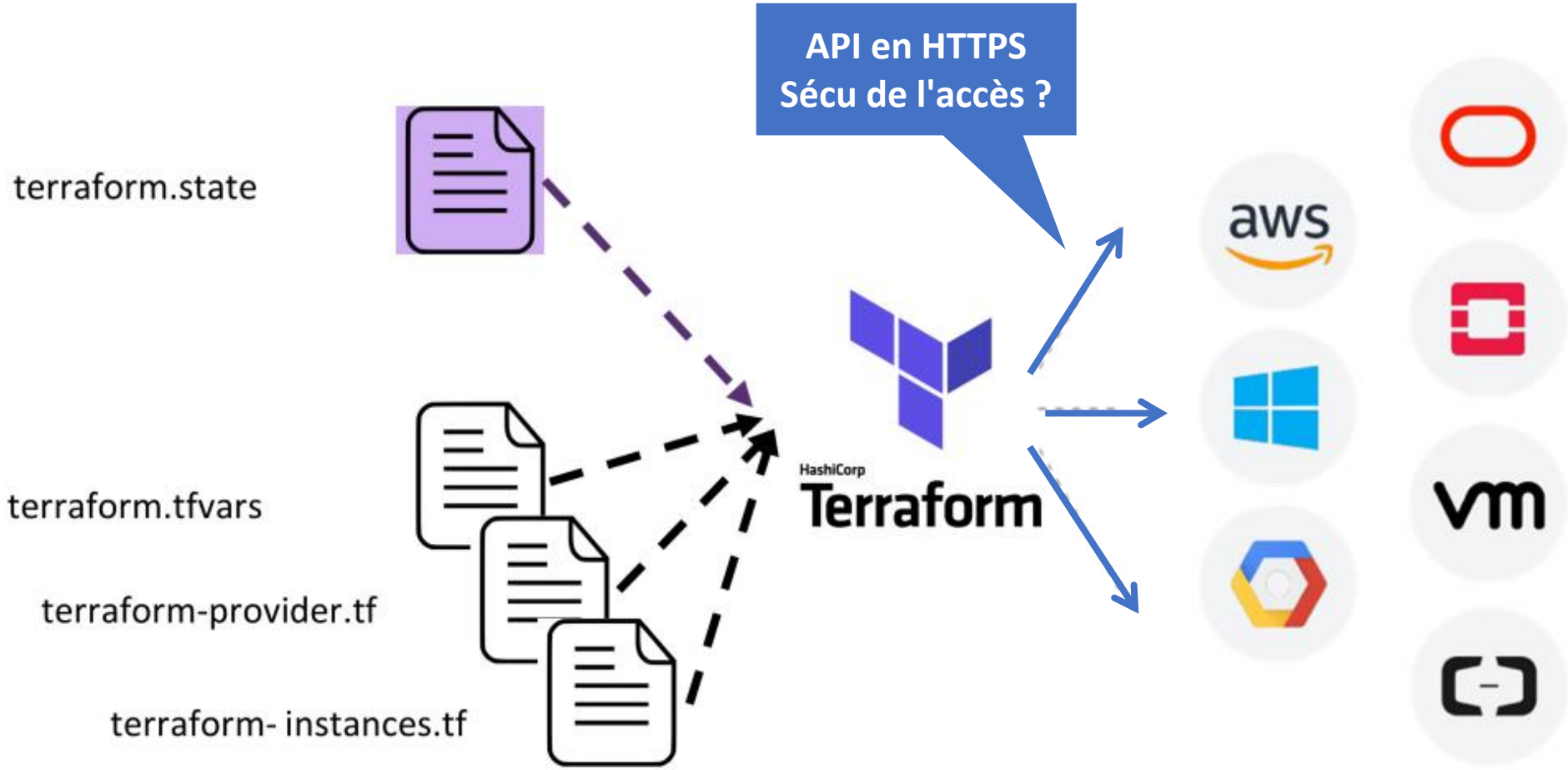
## Risques et vigilances par vis-à-vis de l'infra as code




https://medium.com/formcept/configuration-management-and-continuous-deployment-cd0892dce998

# 8. Infrastructure as Code

## Autre exemple : Terraform pour l'infra virtuelle




https://newstrotteur.fr/2018/12/14/devops101-premiers-pas-sur-terraform-terraform-openstack-ansible/

# 8. Infrastructure as Code

## Infra as Code utilisé comme "Security as Code"

On ~~peut~~ doit (!) tirer partie de l'Infra as Code pour contrôler la sécurité.

On décrit la cible de sécurité à atteindre et l'outil vérifie les éventuels écarts.

Cela réalise un audit automatisé.

Des exemples de solutions : Inspec, Security Monkey (Netflix).

# 8. Infrastructure as Code

## Exemple avec Inspec

```
describe file('/etc/sysconfig/init') do
  its('content') {should match 'umask 027'}
end
```

```
describe port(80) do
  it { should_not be_listening }
end

describe port(443) do
  it { should be_listening }
  its('protocols') {should include 'tcp'}
end
```

```
describe aws_security_group(group_name: linux_servers) do
  its('inbound_rules.first') { should include(from_port: '22',
ip_ranges: ['10.2.17.0/24']) }
end
```

# 9. Containers

# 9. Containers

## Intro sur les containers

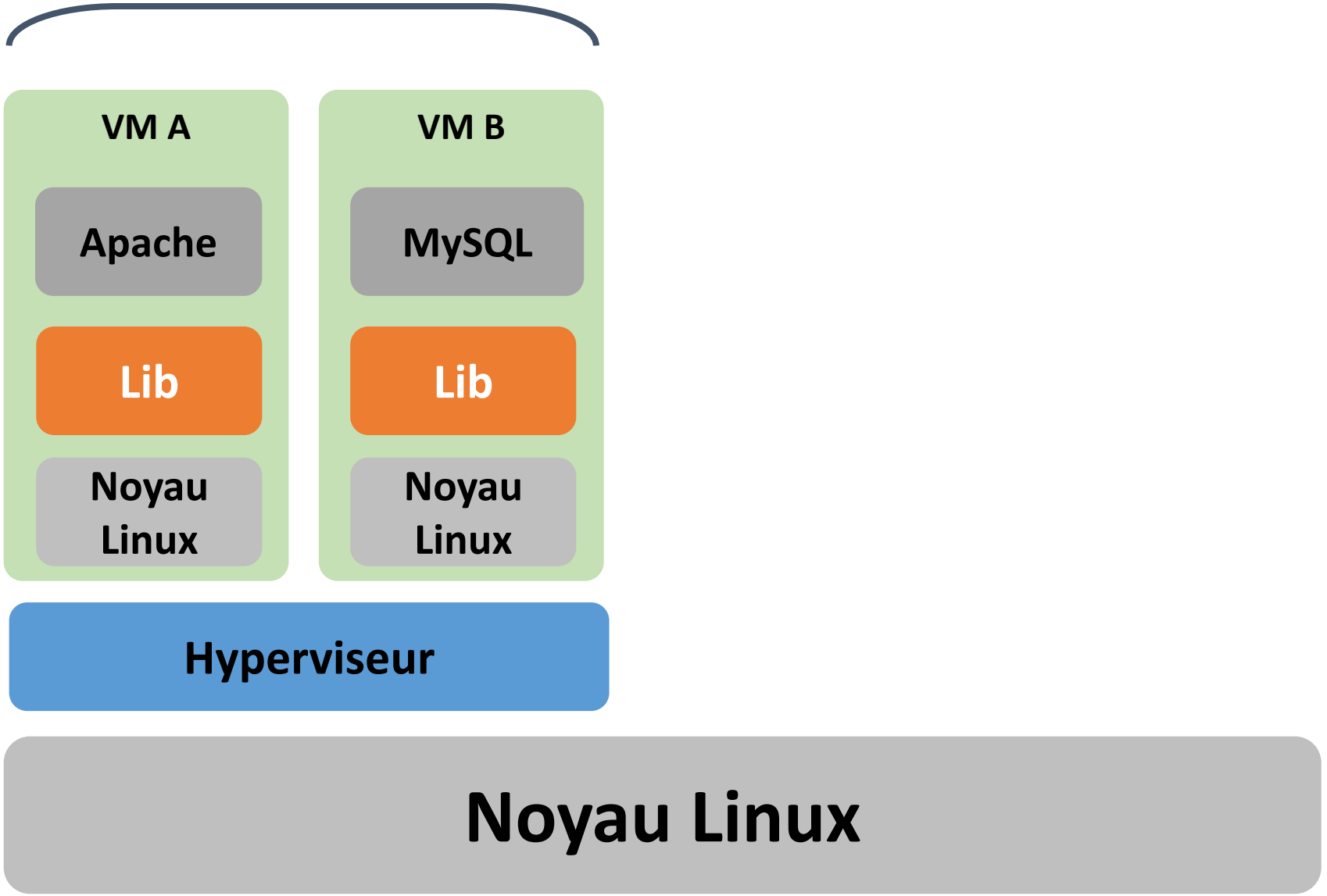
Avec l'émergence de Docker en 2014, les containers sont de plus en plus présents dans l'informatique "moderne".

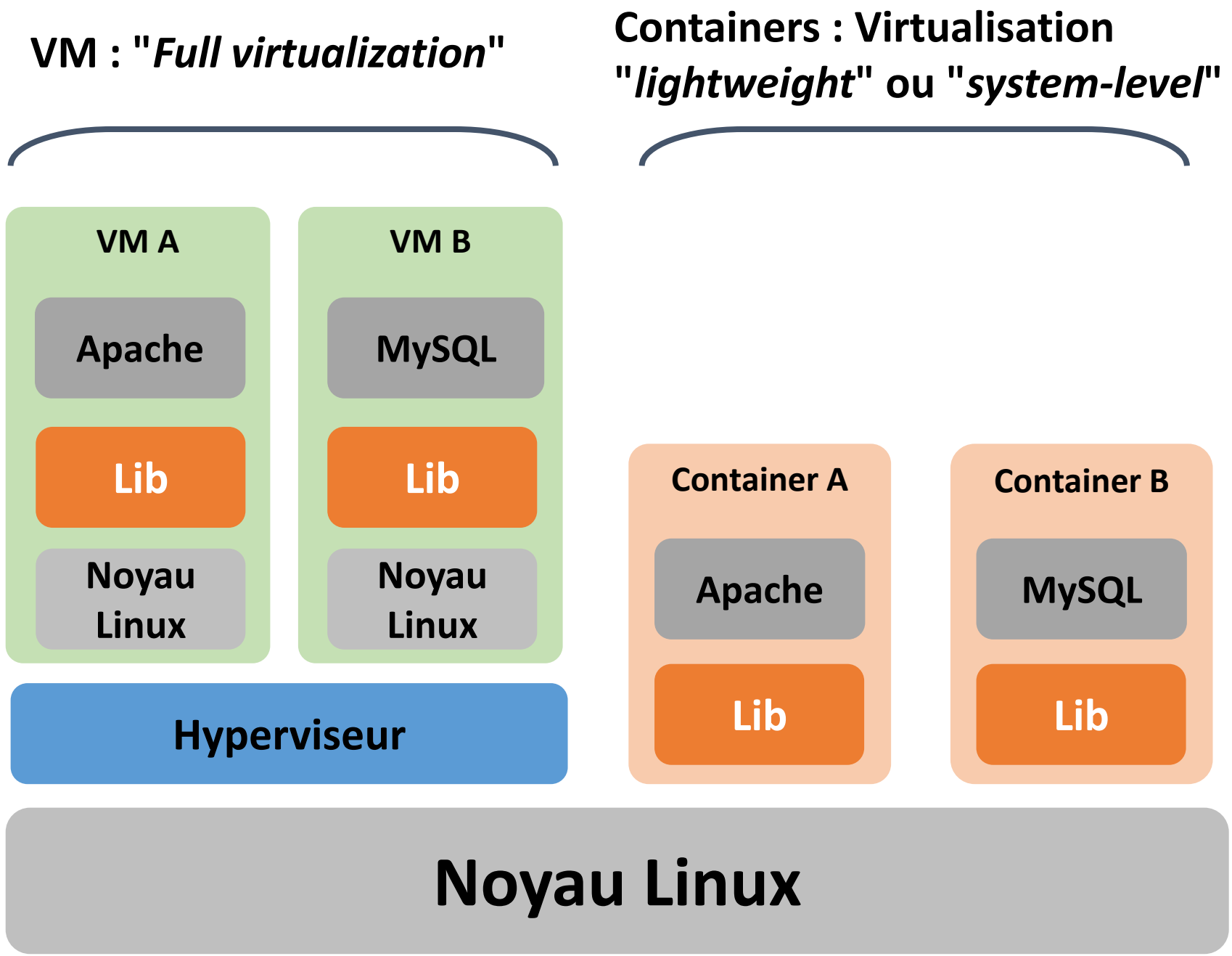
**Les containers n'ont pas été conçus pour la sécurité.**

(Ce qui, de toute façon, est le cas pour la quasi-totalité de l'informatique)

Ils font apparaitre de **nouveaux risques**, mais en même temps de **nouvelles opportunités d'automatisation de la sécurité**.

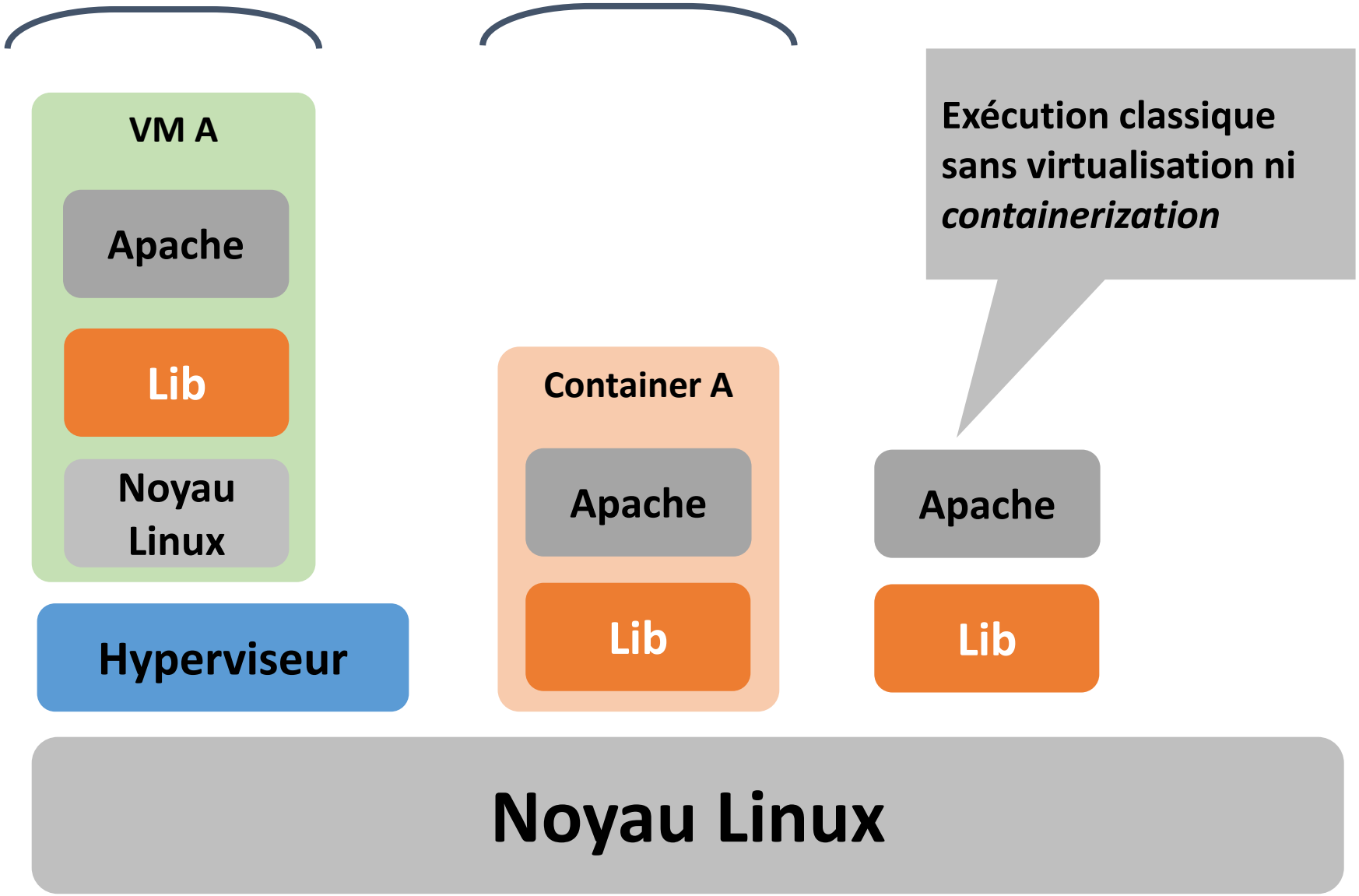
# VM : "*Full virtualization*"





## VM : "*Full virtualization*"

## Containers : Virtualisation "*lightweight*" ou "*system-level*"



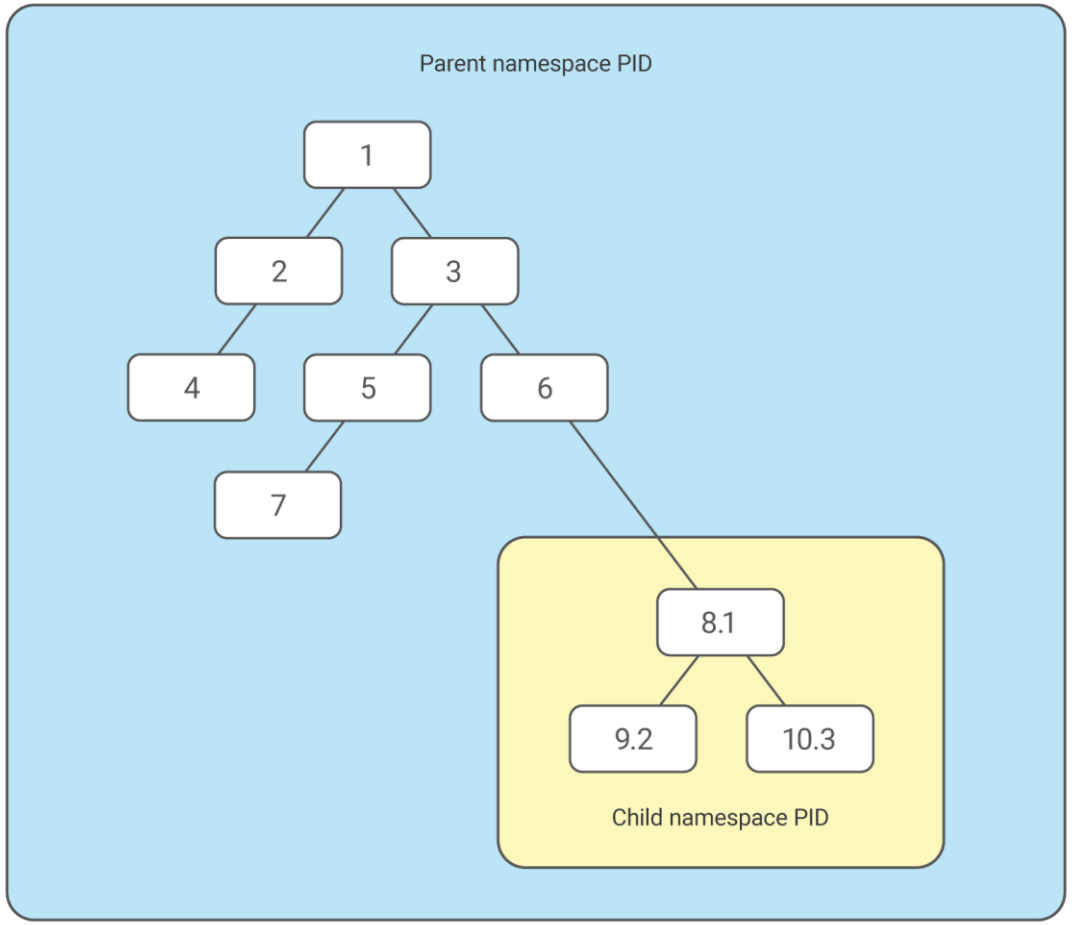
# 9. Containers

## Techniquement

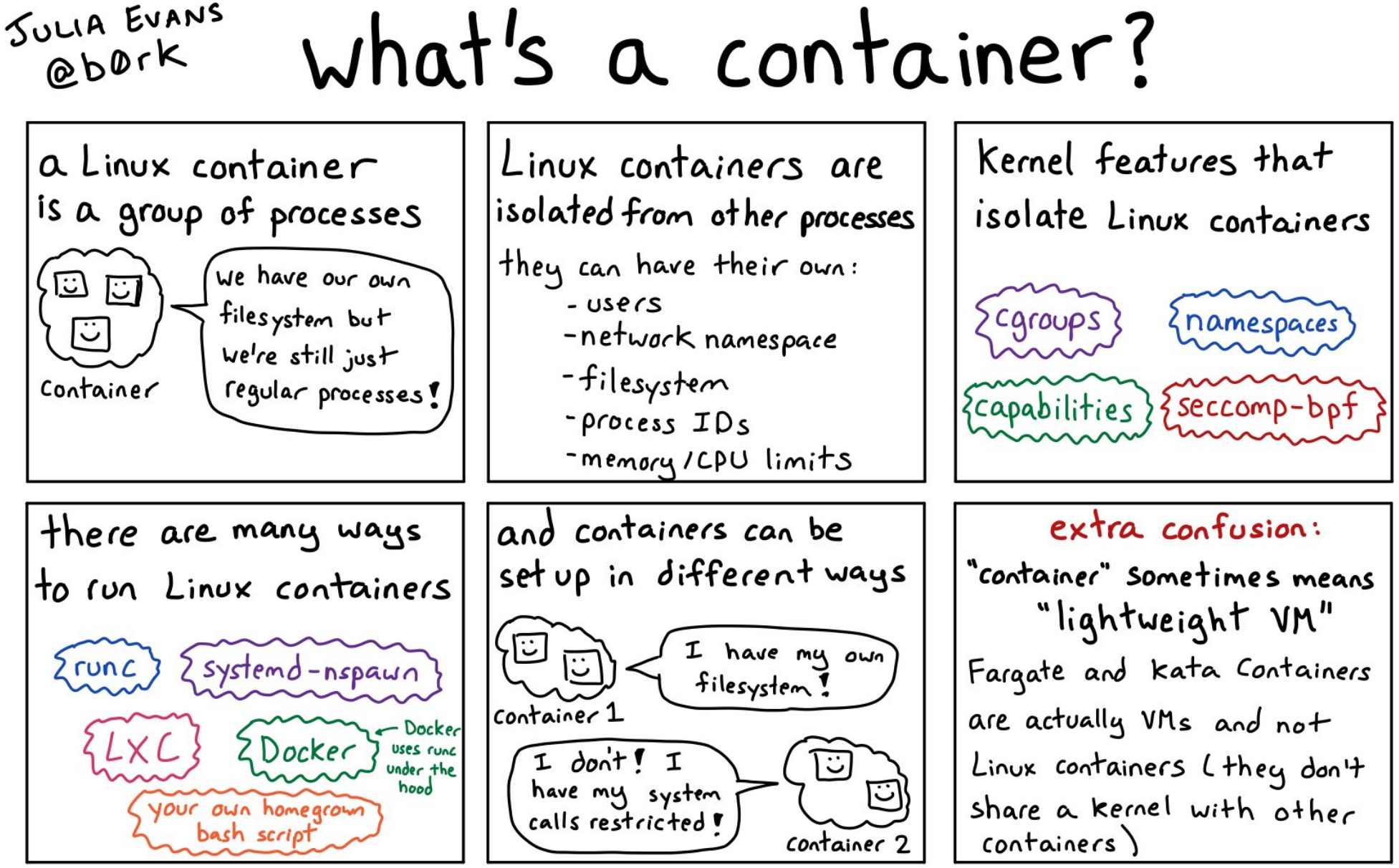
Les containers utilisent les ***namespaces*** du noyau Linux pour le cloisonnement des process.

En plus, ***cgroup*** est utilisé pour appliquer des limites à l'utilisation des ressources (CPU, RAM).

Aussi, il y a une suppression de certaines ***capabilities*** pour limiter les droits à l'intérieur des containers.




https://blog.selectel.com/containerization-mechanisms-namespaces/




De Julia Evans → https://twitter.com/b0rk Recommandation de compte Twitter à suivre !

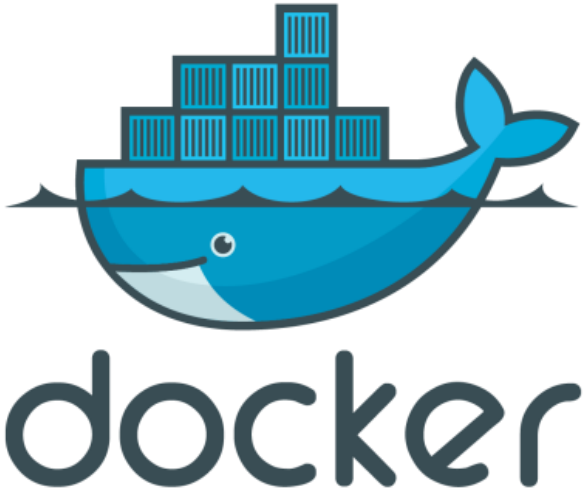
# 9. Containers

## Docker

Docker est la solution de containerisation la plus utilisée.

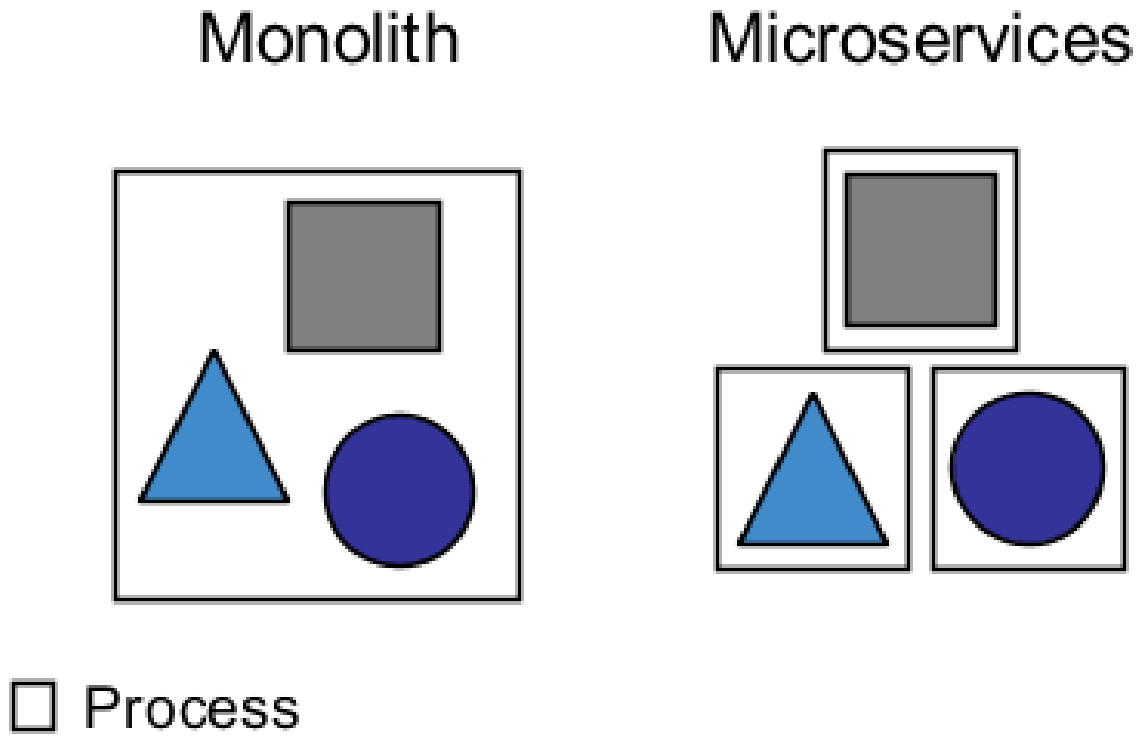
Avant, il y a avait : LXC, V-Server…

Après, il y aura (peut-être) : CRI-O, Podman, Kata, gVisor…



## Pourquoi les containers ?

# Principe de découplage



# 9. Containers

## Apport des containers

**Les containers permettent de réduire l'adhérence entre services.**

→Facilite l'évolution / patching car le couplage entre service est moindre.

**Accélérer le (re)déploiement** et maintenance.

**"Densifier" le déploiement** → Davantage de services sur le même matériel.

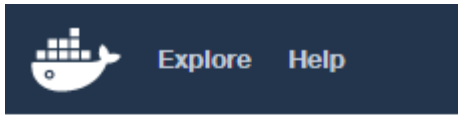
Faciliter l'**orchestration et le scaling**.

# 9. Containers

## Approche image

Récupération des images Docker (images de containers) sur un registre d'image.

```
docker pull centos
```




OFFICIAL REPOSITORY

centos ☆

Last pushed: 4 days ago

OFFICIAL REPOSITORY

mysql ☆

Last pushed: 2 days ago

OFFICIAL REPOSITORY

debian ☆

Last pushed: 15 days ago

# 9. Containers

## Espace quasi mono-process

Exemple d'un container MySQL

```
root@10c64da891d1:/srv# ps -edf
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 Feb10 ?        00:00:00 /bin/bash /usr/bin/mysqld_safe
mysql        355     1  0 Feb10 ?        10:32:22 /usr/sbin/mysqld --basedir=/usr --dat
root         415     0  0 Feb10 ?        00:00:00 /bin/bash
root         428     0  0 Feb10 ?        00:00:00 /bin/bash
root        4585   428  0 17:03 ?        00:00:00 ps -edf
root@10c64da891d1:/srv#
```

# 9. Containers

## Dockerfile pour une construction déterministe

### Dockerfile (tronqué) de Wordpress

```
FROM php:7.3-apache

RUN set -ex; \
curl -o wordpress.tar.gz -fSL "https://wordpress.org/...
\


COPY docker-entrypoint.sh /usr/local/bin/
ENTRYPOINT ["docker-entrypoint.sh"]
```

Image de base

Téléchargement des sources (à chaque build)

Défini le ficher d'entrée à exécuter au lancement du container

Copie (ajoute) un fichier dans le container

# 9. Containers

## Quelques principes

Les containers n'exécutent qu'un seul service (web, base de données, etc.).

→Ainsi, jamais de serveur SSH dans un container (ni aucun agent de monitoring).

Aucune modification (upgrade package ou code) n'est faite dans un container.

Une modification du container implique la reconstruction de l'image du container.

→ Aucune mise à jour sécurité à l'intérieur d'un container.
Il faut reconstruire l'image pour appliquer une MàJ sécurité.

# 9. Containers

## Les risques des containers ?

De nouveaux risques :

- L’image contient-elle des vulnérabilités ?
- L’image récupérée est-t-elle bienveillante ?
- En cas de compromission d'un container, est-ce que l'hôte arrivera à éviter la propagation dans les autres containers ?
- Quelle est l’exposition réseau des containers ?

# 9. Containers

## Maîtriser les risques

### L’image contient-elle des vulnérabilités ?

Utiliser des outils de scan d’images.

Plusieurs opensource:

- Trivy
- Clair
- Anchore

```
ubuntu:xenial-20190515 (ubuntu 16.04)
=====================================
Total: 153 (UNKNOWN: 0, LOW: 11, MEDIUM: 102, HIGH: 40, CRITICAL: 0)

+-------------------+------------------+----------+---------------------------+-
|      LIBRARY      | VULNERABILITY ID | SEVERITY |    INSTALLED VERSION      |
+-------------------+------------------+----------+---------------------------+-
| bsdutils          | CVE-2016-2779    | HIGH     | 2.27.1-6ubuntu3.6         |
|                   |                  |          |                           |
+                   +------------------+----------+                           +-
|                   | CVE-2016-5011    | MEDIUM   |                           |
|                   |                  |          |                           |
|                   |                  |          |                           |
+-------------------+------------------+----------+---------------------------+-
| coreutils         | CVE-2016-2781    | LOW      | 8.25-2ubuntu3~16.04       |
|                   |                  |          |                           |
|                   |                  |          |                           |
+-------------------+------------------+----------+---------------------------+-
| dpkg              | CVE-2017-8283    | HIGH     | 1.18.4ubuntu1.5           |
|                   |                  |          |                           |
|                   |                  |          |                           |
+-------------------+------------------+          +---------------------------+-
| libapparmor1      | CVE-2016-1585    |          | 2.10.95-0ubuntu2.10       |
|                   |                  |          |                           |
|                   |                  |          |                           |
+-------------------+------------------+          +---------------------------+-
| libblkid1         | CVE-2016-2779    |          | 2.27.1-6ubuntu3.6         |
|                   |                  |          |                           |
+                   +------------------+----------+                           +-
```

# 9. Containers

## Maîtriser les risques


ars TECHNICA

BIZ & IT TECH SCIENCE POLICY CARS GAMING & CULTURE

CRYPTOCURRENCY JACKING —

Tesla cloud resources are hacked to run cryptocurrency-mining malware

Crooks find poorly secured access credentials, use them to install stealth miner.

DAN GOODIN - 2/20/2018, 8:21 PM

### L'image récupérée est-t-elle bienveillante ?

Peut-on faire confiance à tout ce que l'on télécharge sur Internet ?

→ Il faut douter des images sur les registres publics (Dockerhub, Quay)

→ Reconstruire ses propres images à partir des images officielles

Le niveau du dessus :

→Signer ses images et interdire l'exécution en prod d'image non-signées.

# 9. Containers

## Maîtriser les risques

**En cas de compromission d'un container, est-ce que l'hôte arrivera à éviter la propagation dans les autres containers ? (1/2)**

→ Audit la machine ou cluster Docker, Kubernetes, OpenShift… sous-jacente pour vérifier les mises à jour et la configuration renforcée.

→ Prévoir un process de patching (sans interruption des services hébergés).

Comme le noyau est partagé entre les containers d'un même hôte, une vulnérabilité noyau peut entrainer la compromission des autres containers de l'hôte, voire du cluster entier.

Scripts de test pour hôtes : https://www.cisecurity.org/benchmark/docker/

## Maîtriser les risques

**En cas de compromission d'un container, est-ce que l'hôte arrivera à éviter la propagation dans les autres containers ? (2/2)**

**Pour éviter une *container-escape* :**

- Mettre à jour le **noyau** !
- **Ne jamais** exécuter le container en mode *privileged.*
- Configurer les ***capabilities*** à retirer au container.
- Bonus : Configurer **SELinux** ou **AppArmor** (restriction d'accès aux fichiers)
- Bonus : configurer seccomp-BPF (filtrage d'appels système)

# 9. Containers

## Maîtriser les risques

### Quelle est l'exposition réseau des containers ?

Un container s'expose de deux façons :

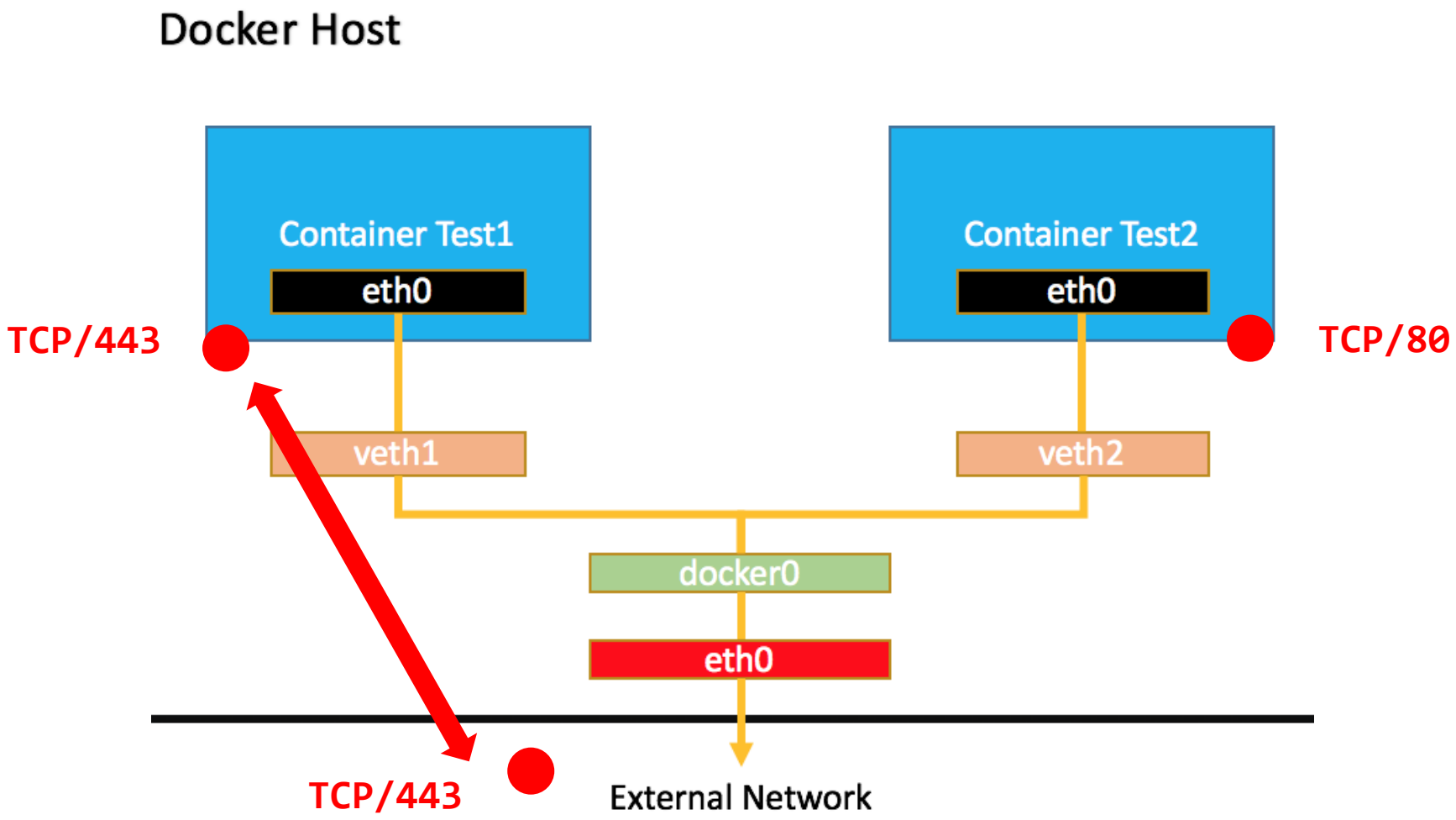
- Dans le Dockerfile : la directive `EXPOSE 80` va exposer le port TCP 80 aux autres containers de l'hôte. Mais pas d'exposition "externe" à l'hôte.
- Au niveau de l'orchestrateur : `--port 443:443` va mapper le port 443 de l'hôte sur le port 443 du container. Ainsi, le container va se retrouver exposé aux machines en réseau avec l'hôte (voire Internet ?).

→ Ne pas exposer les services "internes" sur les ports de l'hôte.

→ Et déjà, est on confiant de l'exposition interne d'un container si son voisin se fait compromettre ?

# 9. Containers

## Schéma réseau




https://medium.com/@xiaopeng163/docker-bridge-networking-deep-dive-3e2e0549e8a0

# 9. Containers

## Service Mesh

Le Service Mesh commence à émerger pour proposer de la sécurisation fine des communications entre containers :

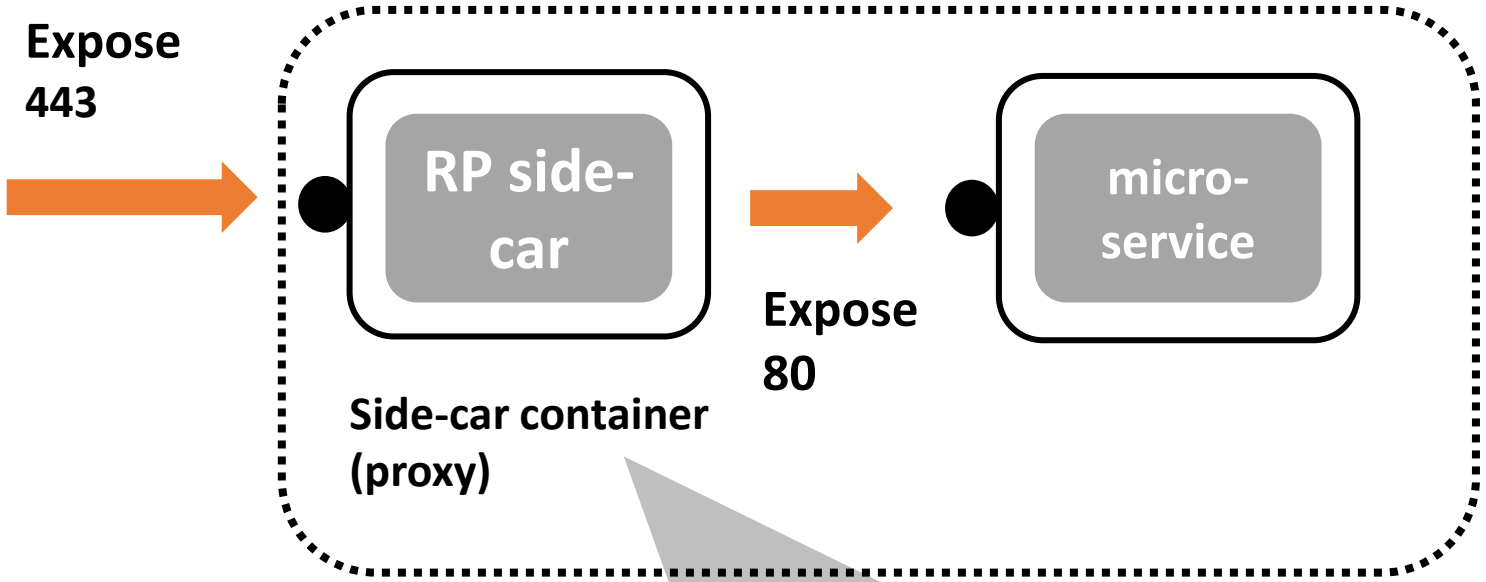
- Notion de cloisonnement réseau au niveau applicatif entre containers.

- Chiffrement, par defaut, de tous les flux entre containers.

Exemple de solutions : Istio, Linkerd…

# 9. Containers

## Notion de side-car containers pour le chiffrement "by default"




**Le side-car container peut :**

- **Terminer et initier les flux TLS.**
- **Appliquer un contrôle d'accès suivant la provenance du flux.**
- **Exporter les métriques du trafic pour des besoins de monitoring.**

# 10. Gestion des secrets

# 10. Gestion des secrets

## Intro

Lorsque que l'on automatise les déploiements, où stocker les mots de passes, clés, tokens, etc ?

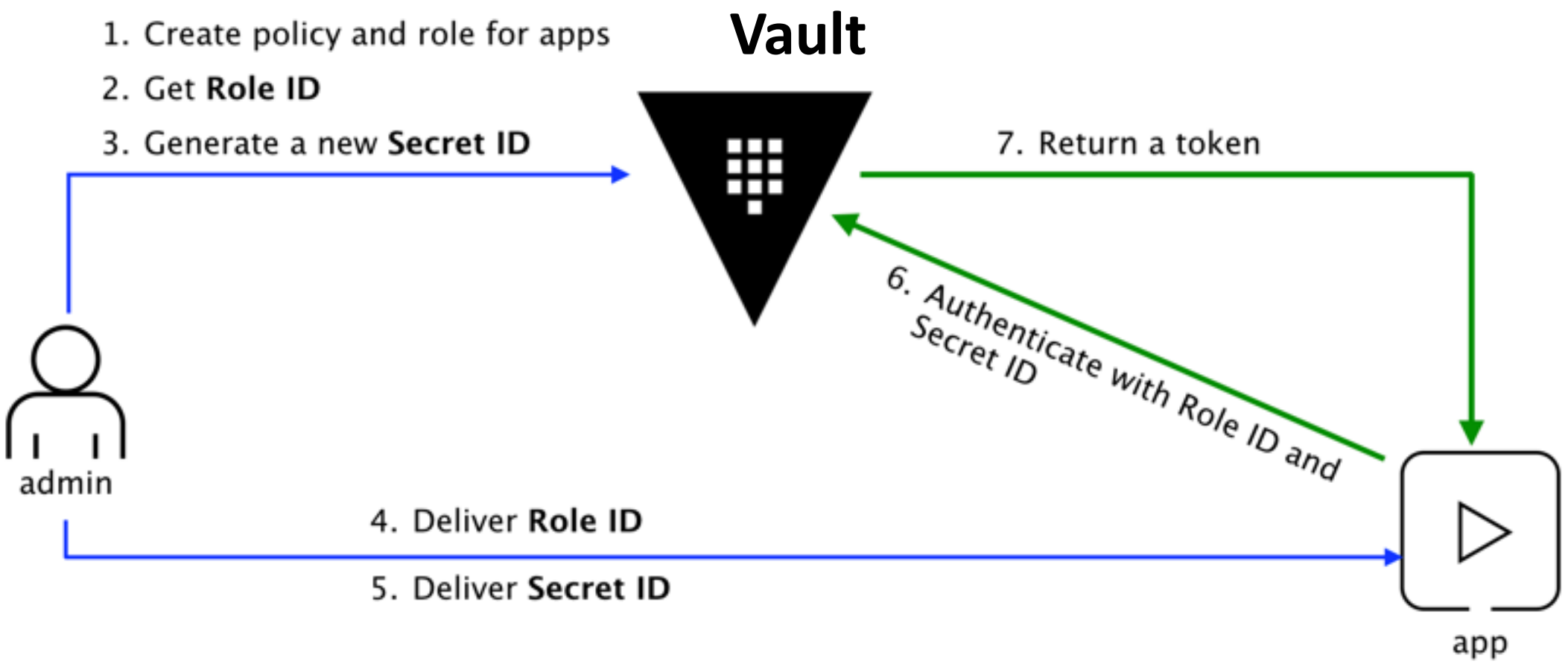
- Dans un playbook stocké dans Git → mauvaise idée !

À la place, un coffre fort à secrets va :

- Générer les secrets (clés, tokens, mots de passe).
- Permettre aux services de venir récupérer leur secret.

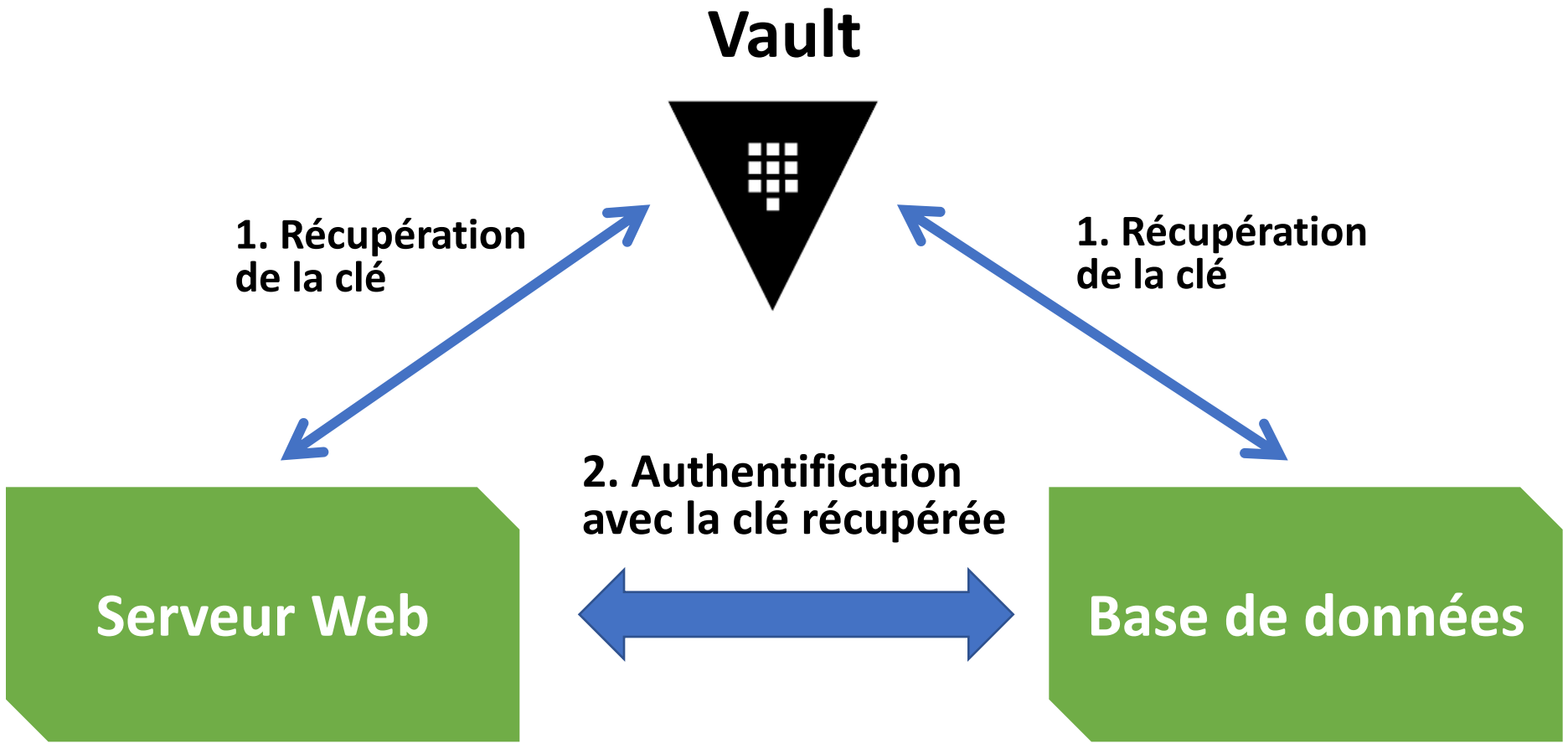
# 10. Gestion des secrets

## Exemple de Vault Hashicorp



# 10. Gestion des secrets

## Ainsi entre deux services



# 10. Gestion des secrets

## D'autres cas d'usage pour la sécurité DevOps

Ce type de Vault permet également d'adresser les cas d'usages suivants :

- Génération de **PKI X.509** (certificats HTTPS).
  - → Générer les certificats HTTPS dynamiquement.
- Génération de **PKI certificats SSH** (qui ne sont pas compatibles X.509).
- Brique de **chiffrement as a service**.

# 11. Pour aller plus loin

# 11. Pour aller plus loin

## Web

Série de 23 articles « Pushing Left, Like a Boss » de Tanya Janca (SheHacksPurple)

https://dev.to/azure/pushing-left-like-a-boss-part-1-4d9i

# 11. Pour aller plus loin

## Comptes Twitter

https://twitter.com/shehackspurple

https://twitter.com/lizrice

https://twitter.com/clintgibler

https://twitter.com/jvehent

https://twitter.com/manicode

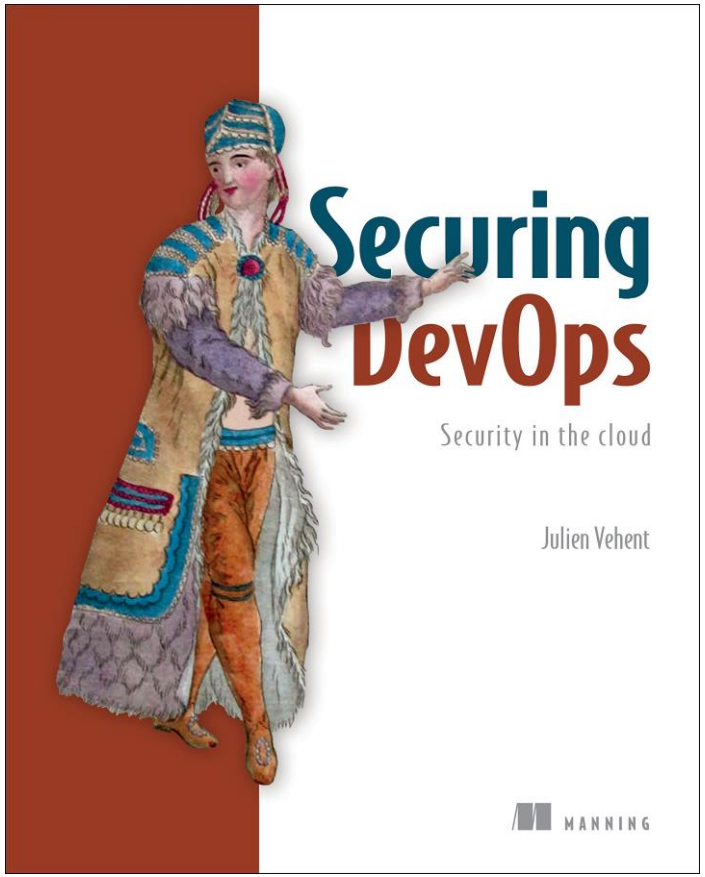
https://twitter.com/TeriRadichel

https://twitter.com/dinodaizovi

https://twitter.com/b0rk

# 11. Pour aller plus loin

## Ouvrages





Laura Bell, Michael Brunton-Spall,
Rich Smith & Jim Bird