# Sécurité des infrastructures de virtualisation

Timothée Ravier

@siosm

https://tim.siosm.fr/cours

5e année cycle ingénieur, filière STI

Option Sécurité des Systèmes Ubiquitaires

2018 – 2019

# Plan global


1 Infrastructures

2 Conteneurs

3 systemd

4 Gestionaires de conteneurs

5 SELinux

6 Conteneurs et sécurité

7 Virtualisation et sécurité

8 Sécurité des infrastructures

# Plan


1 Infrastructures

2 Conteneurs

3 systemd

4 Gestionaires de conteneurs

5 SELinux

6 Conteneurs et sécurité

7 Virtualisation et sécurité

8 Sécurité des infrastructures

# Infrastructures ?

## Infrastructure de type Cloud

Ensemble de **services** qui fournissent aux **utilisateurs** un **environnement** pour déployer des **applications** sous forme de **machines virtuelles**, **conteneurs** ou « **fonctions** ».

## Self service

Chaque utilisateur peut choisir ce dont il a besoin parmis les services proposés (stockage, réseau, base de données, gestion de charge, etc.).

## Responsabilités séparées

Les services fournis par l'infrastructure ne sont pas administrés par les utilisateurs de l'infrastructure.

# Services fournis par une infrastructure

- Compute :
  - Machines virtuelles
  - Conteneurs
  - Serverless (ou « Fonctions »)
- Stockage :
  - Stockage d’images disques (block storage)
  - Stockage d’images de conteneurs (container registry)
  - Stockage d’objets / blocs de données (object storage)
  - Stockage persistant (NFS, etc.)
- Réseau :
  - DNS
  - Load Balancing
  - Enregistrement et découverte de services
  - Routage

# Services fournis par une infrastructure

- Bases de données
- Gestion des secrets
- Gestion des identités
- Gestion de la configuration
- Intégration Continue (CI) :
  - Buildbot
  - Construction de conteneurs
  - Test automatique
- Supervision :
  - Métriques (performance)
  - Inventaire et audit
- Intégration avec du matériel (routeurs, stockage, etc.)
- Intégration avec les services de Cloud providers

# Exemples d’infrastructures

Cloud providers :

- Amazon Web Services,
- Google Cloud,
- IBM Cloud,
- Microsoft Azure,
- Alibaba Cloud,
- Yandex Cloud,
- etc.

# Mettre en place une infrastructure

Nombreux projets proposant un ensemble de logiciels pour mettre en place sa propre infrastructure, dans un environnement public (Cloud provider) ou privé (datacenter) :

- Infrastructure orientée virtualisation :
  - OpenStack ;
  - oVirt ;
  - Apache CloudStack, Eucalyptus, OpenNebula, etc.
- Infrastructure orientée conteneurs :
  - Kubernetes (et dérivés : OpenShift, etc.) ;
  - Hashicorp Nomad ;
  - Mesos, Docker Compose, Cloud Foundry, etc.

# Sécurité des infrastructures de virtualisation

- Sécurité des technologies sur lesquelles se base l'infrastructure :
  - Sécurité des hôtes ;
  - Sécurité des machines virtuelles ;
  - Sécurité des conteneurs.

- Sécurité des services :
  - qui composent l'infrastructure ;
  - qui sont fournis aux utilisateurs par l'infrastructure ;
  - Voir dernier chapitre.

# Plan


1 Infrastructures

2 Conteneurs
- Historique et concepts
- Gestion des ressources (cgroups)
- Isolation (namespaces)
- Capabilities & seccomp

3 systemd

4 Gestionaires de conteneurs

5 SELinux

6 Conteneurs et sécurité

7 Virtualisation et sécurité

8 Sécurité des infrastructures

# Plan


2 Conteneurs

- Historique et concepts
- Gestion des ressources (cgroups)
- Isolation (namespaces)
- Capabilities & seccomp

# Cas d’usage de la virtualisation

- Avantages :
  - Bonne isolation ;
  - Contrôle complet des services et interactions ;
  - Options possibles au niveau du partage du matériel ;
  - Gestion des ressources simple mais fixe.
- Désavantages :
  - Système d’exploitation complet à gérer ;
  - Consommation en ressources importante ;
  - Temps de démarrage ;
  - Duplication des fonctionnalités pour chaque instance : journalisation, contrôle d’accès, etc.

Conclusion : Rarement intéressant de lancer une VM classique pour chaque instance d’une application.

# Cas d’usage des conteneurs (sous Linux)

Objectifs :

- Créer un ensemble de processus indépendant du reste du système ;
- Maîtriser la consommation en ressources système ;
- Isoler ces processus du reste du système ;
- Sans nécessairement utiliser la virtualisation ;
- Démarrage très rapide ;
- Durée de vie plus variable qu’une machine virtuelle.

# Cas d’usage des conteneurs (sous Linux)

En pratique :

- Matériel : abstraction fournie par le noyau souvent suffisante (pas besoin d’émulation (QEMU), même architecture, etc.) ;
- Noyau : la version du noyau n’importe pas pour la plupart des applications ;
- Complexité moindre (moins d’éléments) pour debugger ;
- Partage des services système entre les conteneurs (journalisation, etc.) ;
- Temps de démarrage, taille et consommation en ressource moindre.

# Virtualisation ?

On parle de virtualisation légère, virtualisation au niveau du système d'exploitation ou de conteneurs par opposition à la virtulisation lourde/complète/totale d'un système d'exploitation.

# Historique

- chroot (1982) : exécuter un programme avec un / distinct ;
- FreeBSD Jails (2000) : ajoute des restrictions au niveau des intéractions avec le noyau, le réseau et virtualise le super utilisateur ;
- Linux Vserver (2001) : concept de *context* et fonctionnalités similaire aux jails pour Linux. Patch noyau uniquement, support très limité.
- Solaris Zones (2004) : concept de *system call translation* ;
- OpenVZ (2005) : patch noyau Linux uniquement, support limité. Certaines fonctionnalités ont été intégrées upstream (CRIU).

**Attention !**

Pas de concept de conteneur dans le noyau Linux !

Il faut combiner « manuellement » :

- la gestion des ressources et de l'accès à certains périphériques avec les cgroups ;
- l'isolation (modulable) avec les namespaces ;
- les restrictions de permissions avec les capabilities, les filtres seccomp et les Linux Security Modules (SELinux / AppArmor).

# Plan


2 Conteneurs

- Historique et concepts
- Gestion des ressources (cgroups)
  - Gestion statique des ressources
  - Gestion dynamique : cgroups
- Isolation (namespaces)
- Capabilities & seccomp

# rlimits : resource limits

- Limite les ressources disponibles pour l'ensemble des processus d'un utilisateur ou d'un groupe ;
- Affichage / manipulation des limites avec `ulimit` ;
- Deux types de limites : `hard` et `soft` ;
- Limite `soft` :
  - limite appliquée par le noyau ;
  - définissable entre 0 et <limite `hard`>.
- `CAP_SYS_RESOURCE` pour augmenter les limites `hard` ;
- Réduction des limites `hard` irréversibles pour `!root` ;
- Limites initiales appliquées lors du login par PAM, configurées dans `/etc/security/limits.conf`.

# rlimits : resource limits

- cpu time (seconds) : unlimited
- file size (blocks) : unlimited
- data seg size (kbytes) : unlimited
- stack size (kbytes) : 8192
- core file size (blocks) : 0
- resident set size (kbytes) : unlimited
- processes : 1024
- file descriptors : 1024
- locked-in-memory size (kbytes) : 64
- address space (kbytes) : unlimited
- file locks : unlimited
- pending signals : 22949
- bytes in POSIX msg queues : 819200
- max nice : 0
- max rt priority : 0

# Quota disque

- Limite les ressources disque par `user` et par `group` ;
- Limite `soft` & `hard` par système de fichier ;
- La limite `hard` ne peut pas être dépassée ;
- La limite `soft` peut être dépassée pour un temps donné ;
- Une fois ce temps dépassée, la limite `soft` devient `hard` ;
- L'utilisateur doit libérer de la place pour redescendre au dessous de la limite `soft` avant de pouvoir écrire à nouveau des données ;
- Quota disque par « projets » avec XFS [1].

# Espace disque réservé pour les processus privilégiés

- Il est possible de réserver de l'espace sur un système de fichier pour permettre aux processus privilégiés (syslog / journald) de continuer à fonctionner lorsque le disque est presque plein ;
- `$ tune2fs -m reserved-blocks-percentage`
- Valeur par défaut : 5 % de la taille du système de fichier.

# L’overcommit

- Chaque processus a l’impression d’être le seul à utiliser le matériel ;
- On lui propose donc plus de ressources que ce qui est réellement disponible sur la machine ;
- Exemple : l’espace d’adressage virtuel
  - $2^{32} \approx$ 4Go en 32bits. Pour l’instant $2^{48}$ en 64 bits ;
  - `/proc/sys/vm/{overcommit_memory,overcommit_ratio}` ;
  - `Documentation/vm/overcommit-accounting`.
- En tenir compte lorsque l’on atteint les limites.

# Priorités, IO, et CPU

- nice level : Défini la priorité d'accès aux ressources CPU pour un processus ;
- ionice : Défini la priorité d'accès aux périphériques pour un processus ;
- cpu affinity : Oblige un processus à utiliser des cœurs définis.

# Limites du modèle statique

- Flexibilité très limitée ;
- Il faut gérer « à la main » les changements de limite : pas de répartition automatique entre les utilisateurs ou les processus ;
- Peu adapté au modèle où l'on va avoir beaucoup de machines virtuelles ou conteneurs utilisées par un seul utilisateur ;
- Ou à des services constitués de plusieurs processus que l'on souhaite gérer comme un tout.

# cgroups

- Introduit dans le noyau 2.6.24 par des ingénieurs travaillant chez Google ;
- Permets de regrouper des processus en leur associant un label ;
- Organisation hiérarchique ;
- Les processus fils héritent du cgroup du père ;
- Représenté par un système de fichier virtuel : `/sys/fs/cgroup/`.

# cgroups : `controllers`

- Configuration d'un ou plusieurs `controllers` pour chaque groupe de tâches ;
- Imposent des limites ou contraintes communes à l'ensemble des processus de chaque groupe.
- *cgroups(7)* contient la liste complète ainsi que les versions du noyau à partir desquelles chaque `controller` est disponible ;
- Liste des `controllers` disponibles sur un système : `/proc/cgroups`.

# cgroups : `controllers` disponibles

- cpu & cpuacct : mesure et limite l'utilisation CPU ;
- cpusets : répartition sur les CPUs / nœuds NUMA ;
- memory : mesure et limite l'utilisation de la mémoire ;
- blkio : mesure et limite l'utilisation des périphériques de type block ;
- net_cls & net_prio : classe et définit des priorités pour les paquets réseaux ;
- devices : restreint l'accès aux devices disponibles ;
- freezer : pause l'exécution des processus du groupe ;
- perf_event : groupes spécifiques au sous-système *perf* ;
- rdma : limite les accès RDMA et InfiniBand ;
- hugetlb : limite l'utilisation des Huge Page.
- pids : limite le nombre de PID disponibles.

# cgroups v1 & v2

Deux versions des cgroups :

- cgroups v1 :
  - chaque `controller` a sa propre hiérarchie ;
  - pose des problèmes de gestion, performance, complexité ;
  - diverses limites liées au design.
- cgroups v2 :
  - une seule hiérarchie commune à tous les `controllers` [3] ;
  - certains `controllers` pas (encore) disponibles ;
  - configuration différente pour certains `controllers`.

# cgroups v2 : `controllers` disponibles

- cpu : mesure et limite l'utilisation CPU ;
- cpusets : répartition sur les CPUs / nœuds NUMA ;
- memory : mesure et limite l'utilisation de la mémoire ;
- io : mesure et limite l'utilisation des périphériques de type block ;
- perf_event : groupes spécifiques au sous-système *perf* ;
- rdma : limite les accès RDMA et InfiniBand ;
- pids : limite le nombre de PID disponibles.

# Plan


2 Conteneurs

- Historique et concepts
- Gestion des ressources (cgroups)
- Isolation (namespaces)
- Capabilities & seccomp

# Isolation avec les namespaces I

Principe général :

- Crée un ou plusieurs espace de nom (namespaces ou ns) pour séparer le contexte d'exécution d'un processus par rapport aux autres ;
- Isole les objets créés dans un namespace vis à vis des autres namespaces ;
- Lorsque tous les processus d'un namespace ont terminé leur exécution, les objets restant sont détruits ou renvoyés dans le namespace parent.

# Isolation avec les namespaces II

- Fonctionnalité ancienne (début dans la branche 2.4 !) mais étendue récemment ;
- Au départ ajouté comme *flags* à l'appel système `clone` :

```
int clone(int (*fn)(void *), void *child_stack,
          int flags, void *arg, ...
          /* pid_t *ptid, void *newtls, pid_t *ctid */ );
```

- CLONE_NEWNET, CLONE_NEWNS, CLONE_NEWPID, CLONE_NEWUTS, CLONE_NEWIPC, CLONE_NEWUSER, CLONE_NEWCGROUP...
- Nouveaux appels systèmes pour modifier le processus courant :

```
int unshare(int flags);
int setns(int fd, int nstype);
```

- Voir *namespaces(7)*, *unshare(2)*, *clone(2)*, *setns(2)*.

# CLONE_NEWNS

Crée un namespace pour les points de montage :

- Présent depuis le noyau 2.4.19 (2002) ;
- Permet au fils de monter/démonter des systèmes de fichier sans impacter le père ;
- Nécessite `CAP_SYS_ADMIN`.

# CLONE_NEWUTS

Crée un namespace UTS :

- Présent depuis le noyau 2.6.19 (2006) ;
- Contient le domain name et le host name ;
- Permets au fils de changer ces informations sans impacter le père ;
- Nécessite `CAP_SYS_ADMIN`.

# CLONE_NEWIPC

Crée un namespace pour les IPC :

- Introduit dans le noyau 2.6.19, complété dans le 2.6.30 (2009) ;
- Vue isolée des IPC System V : seuls les processus de ce namespace peuvent accéder à ces objets ;
- Destruction des objets à la fin du namespace ;
- Nécessite `CAP_SYS_ADMIN`.

# CLONE_NEWNET

Crée un namespace pour les interface réseaux :

- Introduit dans le noyau 2.6.24, complété dans le 2.6.29 (2009) ;
- Le fils dispose de sa propre stack réseau : interfaces, stacks IPv4/IPv6, tables de routage IP, règles de parfeu, sockets...
- Une interface physique ne peut être que dans un seul namespace à la fois : rattachée au namespace initial à la destruction ;
- Il faut donc créer des interfaces virtuelles pour pouvoir les utiliser dans les namespaces ;
- Nécessite `CAP_SYS_ADMIN`.

# CLONE_NEWPID

Crée un namespace pour les PID :

- Présent depuis le noyau 2.6.24 (2008) ;
- Les PIDs recommencent à 1 dans ce namespace ;
- Les processus du namespace parent voient les processus du namespace fils avec des PIDs différents ;
- Les processus dans un namespace ne peuvent voir que ceux du même namespace et des namespaces créés par des processus de leur namespace ;
- Nécessite `CAP_SYS_ADMIN`.

# CLONE_NEWUSER

Crée un namespace pour les UID/GID :

- Introduit dans le noyau 2.6.24, finalisé dans le 3.8 (2013) ;
- Associe un ensemble d'UID/GID dans le nouveau namespace au couple UID/GID initial ;
- `root` dans un user namespace sans l'être en dehors ;
- Le noyau s'assure qu'il n'y as pas d'élévation de privilèges par rapport au couple UID/GID initial ;
- Nécessite la mise en place d'une correspondance entre les UIDs à l'intérieur et à l'extérieur du user namespace ;
- Mode *Unpriviliged* disponible depuis le 3.8. Sinon `CAP_SYS_ADMIN`, `CAP_SETUID` and `CAP_SETGID`. Comportement variable suivant les distributions.

# CLONE_NEWCGROUP

Crée un namespace pour les hiérarchies cgroups :

- Introduit dans le noyau 4.6 (2016) ;
- Abstraction des hiérarchies cgroups pour les conteneurs ;
- Non privilégié.

# Outils en ligne de commande

- `unshare` : lancer un programme en utilisant de nouveaux namespaces ;
- `nsenter` : lancer un programme avec les namespaces d'un programme existant ;
- Visualiser les namespaces associés à un programme :

```
$ ls -l /proc/1006/ns/*
... /proc/1006/ns/cgroup -> 'cgroup:[4026531835]'
... /proc/1006/ns/ipc -> 'ipc:[4026531839]'
... /proc/1006/ns/mnt -> 'mnt:[4026531840]'
... /proc/1006/ns/net -> 'net:[4026531993]'
... /proc/1006/ns/pid -> 'pid:[4026531836]'
... /proc/1006/ns/pid_for_children -> 'pid:[4026531836]'
... /proc/1006/ns/user -> 'user:[4026531837]'
... /proc/1006/ns/uts -> 'uts:[4026531838]'
```

# Plan


2 Conteneurs

- Historique et concepts
- Gestion des ressources (cgroups)
- Isolation (namespaces)
- Capabilities & seccomp
  - Capabilities
  - seccomp

# *Capabilities* I

- L'utilisateur `root` est tout puissant sur un système Linux ;
- Tous les autres utilisateurs sont non privilégiés par défaut ;
- Les *capabilities* Linux : donner une partie des privilèges de `root` à des programmes non privilégiés ;
- Exemple :
  - `CAP_NET_BIND_SERVICE` : Bind un socket sur un port < 1024 ;
  - `CAP_NET_ADMIN` : Configurer les interfaces réseaux ;
  - `CAP_SETUID` : Définir arbitrairement les UIDs d'un processus ;
  - `CAP_SYS_ADMIN` : Effectuer un grand nombre d'opération privilégiées ;
  - Voir **capabilities(7)** pour la liste complète.
- Peuvent aussi être associées à des exécutables pour être ainsi transmises aux processus résultant.

# *Capabilities* II

- Comme pour les UID/GID, chaque *thread* possède son propre ensemble de *capabilities* ;
- Leur état est stocké dans plusieurs variables (nommés *sets*) :
  - Effective : le set utilisé pour les contrôles dans le noyau ;
  - Permitted : limite actuelle des *capabilities* disponibles ;
  - Inheritable : préservées lors d'un appel à `execve` et conservées si le fichier exécuté les possède ;
  - Ambient : préservées lors d'un appel à `execve` dans tous les cas.
- Capabilities Bounding Set : limite définitive des *capabilities* disponibles pour un *thread* ;
- Ensemble de régles complexes pour déterminer quelles *capabilities* sont conservées / perdues / transmises lors des appels à `execve`, `setuid`, ...
- Voir **capabilities(7)**.

# *Capabilities* & `root`

- De nombreuses *capabilities* sont en pratique équivalente à `root` :
  - `CAP_SYS_ADMIN` : the new root [41] ;
  - False Boundaries and Arbitrary Code Execution [40].

# seccomp & seccomp-bpf

- seccomp : mode strict qui n'autorise un processus à ne faire usage qu'aux appels système `read`, `write` et `_exit` ;
- seccomp-bpf : extension du modèle initial qui filtre les appels système disponibles pour un processus ;
- Filtre décrit sous forme de programme BPF (*Berkeley Packet Filter*) ;
- Si le processus utilise un appel système filtré, le noyau peut retourner un code d'erreur, d'envoyer un signal ou tuer le processus ;
- En pratique : utilisation de la libseccomp, des options de systemd ou du gestionnaire de conteneurs ;
- Possibilité de créer des blacklists / whitelists.

# Plan


1 Infrastructures

2 Conteneurs

3 systemd
- Historique
- Fonctionnement
- journald
- logind, networkd, timesyncd, resolved, etc.
- Sécurité

4 Gestionaires de conteneurs

5 SELinux

6 Conteneurs et sécurité

7 Virtualisation et sécurité

8 Sécurité des infrastructures

# Plan


3 systemd

- Historique
- Fonctionnement
- journald
- logind, networkd, timesyncd, resolved, etc.
- Sécurité

# Histoire : SystemV et sysinit

- Runlevels (`/etc/inittab`) ;
- Script shell pour chaque service (`/etc/init.d/nginx`) ;
- Beaucoup de répétitions, peu de mutualisation, beaucoup de hacks ;
- Démarrage séquentiel, arrêt séquentiel ;
- Chaque distribution a ses propres scripts pour chaque service ;

# Histoire : SystemV et sysinit

- Difficile de maintenir des modifications dans la durée ;
- Aucune information sur l'état du système ;
- Pas de surveillance de l'état du système ;
- Difficile de suivre les processus dans la durée (`/var/run/nginx.pid` ?) ;
- Le processus PID 1 ne sert plus à rien une fois le système démarré.

# Histoire : Ubuntu : Upstart

- Démarrage des service à la suite d'événements ;
- La carte bluetooth est disponible $\Rightarrow$ démarre le service bluetooth ;
- Le réseau est disponible $\Rightarrow$ montage des disques distants ;
- Difficile d'empêcher un service de démarrer automatiquement ;
- Meilleur que SysV, mais pas assez puissant.

# Histoire : Mac OS X : launchd

- Introduit avec Mac OS X 10.4 ;
- Démarre le système et gère les services ;
- Démarrage des services en parallèle et à la demande ;
- Démon launchd (PID 1) et « télécommande » launchctl pour gérer le les services ;
- Gestion des daemons par utilisateur possible ;
- Configuration en XML.

# Histoire : et les autres…

D’autres systèmes sont présentés dans l’article [12]. Ils reposent généralement sur un langage de script :

- OpenRC (Gentoo) ;
- rcNG (NetBSD, FreeBSD et DragonFlyBSD) ;
- runit ;
- Service Management Facility (SMF, OpenIndiana).

# systemd

- Première version annoncée le 30 mars 2010 ;
- Développeurs principaux : Lennart Poettering, Kay Sievers, Harald Hoyer, Daniel Mack, Tom Gundersen, David Herrmann ;
- Dernière version : 240, sortie le 21 décembre 2018 ;
- Adopté comme `init` par défaut par pratiquement toutes les distributions majeures (Fedora, Mageia, openSUSE, Arch Linux, CoreOS, RHEL, SUSE Linux Enterprise Server, Ubuntu, Debian).

# systemd

- Au début : juste un remplaçant d'`init` ;
- Désormais : ensemble de blocs de base pour construire un système Linux.
- Beaucoup de rumeurs et fausses informations circulent à propos de systemd [13] ;
- Adapté à tous les environnements : Distributions embarquées, voitures, téléphones, tablettes, supercalculateurs, desktops...

# Plan


3 systemd

- Historique
- Fonctionnement
- journald
- logind, networkd, timesyncd, resolved, etc.
- Sécurité

# Fonctionnalités

- Gestionnaire des **services** et du **système** : vue globale ;
- Contrôle des « **units** » plutôt que des démons ou processus ;
- Gestion des dépendances entre les units ;
- Suivi des processus qui composent chaque unit :
  - Correspondance unit $\Leftrightarrow$ processus ;
  - Arrêt **propre** et **maîtrisé** des units.
- Temps de démarrage minimal ;
- Debuggable : Reproductibilité, récupération de tous les logs du système.
- Meilleure interface utilisateur (`systemctl`) ;
- « Compatible » avec l'existant.

# Démons principaux

## Système

- systemd (PID 1), udevd, journald, logind

# Démons principaux

## Système

- systemd (PID 1), udevd, journald, logind

## Utilisateur, lancé à la demande

- systemd --user

# Démons principaux

## Système

- systemd (PID 1), udevd, journald, logind

## Utilisateur, lancé à la demande

- systemd --user

## Système, lancés à la demande

- localed, hostnamed, timedated

# Démons principaux

## Système

- systemd (PID 1), udevd, journald, logind

## Utilisateur, lancé à la demande

- systemd --user

## Système, lancés à la demande

- localed, hostnamed, timedated

## Système, optionnels, lancés à la demande

- timesyncd, machined, networkd, resolved, socket-proxyd

# Démons principaux

## Système

- **systemd (PID 1)**, **udevd**, **journald**, **logind**

## Utilisateur, lancé à la demande

- systemd --user

## Système, lancés à la demande

- localed, hostnamed, timedated

## Système, optionnels, lancés à la demande

- timesyncd, machined, networkd, resolved, socket-proxyd

# Init : PID 1

- « Premier » processus lancé par le noyau ;
- Succède dans le cas général à l'*initrd* ;
- Les partitions `/` et `/usr` ont été montées par l'*initrd* ;
- Chargé de démarrer le reste du système ;
- Parent de tous les processus orphelins.

# Init : **systemd** PID 1

- Spécialiste du **lancement de processus** :
  - **Quoi ?** service nginx, mysql...
  - **Comment ?** UID, GID, CWD, capabilities, env...
  - **Quand ?** au démarrage, gestion des dépendances, activation suite à un timer, une socket, un message D-Bus...
  - **Qui ?** Logs.
- et de leur **suivi** :
  - **Qui ?** Quels processus font parti de mon service ?
  - **Status :** watchdogs, socket de notification ;
  - **Redémarrage :** en cas d'échec, à la réception d'un SIGSEGV ;
  - **Arrêt propre :** aucun processus oublié.

# cgroups

- systemd place les **processus** de chaque **unit** dans un **cgroup distinct** ;
- systemd « **seul** » **gestionnaire** des cgroups ;
- Changement de granularité : processus $\Rightarrow$ service (cgroup) ;
- Réponse facile aux questions :
  - A quel service appartient ce processus ?
  - Quels processus font partie de ce service ?

# cgroups : systemd-cgls

```
Control group /:
-.slice
├─init.scope
│ └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 24
├─machine.slice
│ └─machine-qemu\x2dfedora22.scope
│   └─11026 /usr/sbin/qemu-system-x86_64 -name fedora22 -S -machine pc-i440fx-2.3,accel=kvm...
├─system.slice
│ ├─dbus.service
│ │ └─569 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation
│ ├─systemd-journald.service
│ │ └─309 /usr/lib/systemd/systemd-journald
│ ├─systemd-udevd.service
│ │ └─339 /usr/lib/systemd/systemd-udevd
└─user.slice
  └─user-1000.slice
    ├─user@1000.service
    │ ├─pulseaudio.service
    │ │ ├─1446 /usr/bin/pulseaudio --daemonize=no
    │ │ └─1466 /usr/lib/pulse/gconf-helper
    │ ├─gpg-agent.service
    │ │ └─1304 /usr/bin/gpg-agent --daemon --enable-ssh-support --pinentry-program /usr/bin/pinentry-qt
    │ ├─init.scope
    │ │ ├─1289 /usr/lib/systemd/systemd --user
    │ │ └─1290 (sd-pam)
    └─session-c2.scope
      ├─ 1411 kwin_x11
      ├─ 6255 /usr/bin/tmux
      ├─19664 /usr/bin/dolphin
      ├─24500 /usr/bin/thunderbird
      ├─27086 /usr/bin/zsh
```

# Units **systemd**

**11 types d'units :**

- **nginx.service**
- **multi-user.target**
- sshd.socket
- systemd-tmpfiles-clean.timer

- -.**mount**, home.**mount**
- acpid.**path**
- system.**slice**, user-1000.**slice**
- session-1.**scope**

- sys-devices-virtual-block-dm\x2d0.**device**
- proc-sys-fs-binfmt_misc.**automount**
- dev-mapper-system\x2dswap.**swap**

# Units : Emplacements

- `(/usr)/lib/systemd/system` : Fournies par la distribution (lecture seule) ;
- `/run/systemd/system` : Générées à l'exécution, non persistantes ;
- `/etc/systemd/system` : Crées et modifiées par l'administrateur.

- Ordre de priorité : `/etc/` $\rightarrow$ `/run/` $\rightarrow$ `/usr/lib/`

- Liste des units :
  - installées : `systemctl list-unit-files`
  - chargées : `systemctl list-units`

# Units : **httpd.service**

```
# systemctl cat httpd.service
```

```
[Unit]
Description=Apache Web Server
After=network.target remote-fs.target nss-lookup.target

[Service]
Type=forking
PIDFile=/run/httpd/httpd.pid

ExecStart=/usr/bin/apachectl start
ExecStop=/usr/bin/apachectl graceful-stop
ExecReload=/usr/bin/apachectl graceful

PrivateTmp=true
LimitNOFILE=infinity

[Install]
WantedBy=multi-user.target
```

# Units : **httpd.service**

```
# systemctl show -p Description httpd.service
```

```
[Unit]
Description=Apache Web Server
```

# Units : **httpd.service**

```
# systemctl start httpd.service
```

```
1 [Unit]
2 Description=Apache Web Server
3
4 [Service]
5 Type=forking
6 PIDFile=/run/httpd/httpd.pid
7
8 ExecStart=/usr/bin/apachectl start
```

# Units : **httpd.service**

```
# systemctl stop httpd.service
```

```
1 [Unit]
2 Description=Apache Web Server
3
4 [Service]
5 Type=forking
6 PIDFile=/run/httpd/httpd.pid
7
8 ExecStart=/usr/bin/apachectl start
9 ExecStop=/usr/bin/apachectl graceful-stop
```

```
# systemctl kill -signal=SIGTERM httpd.service
```

# Units : **httpd.service**

```
# systemctl reload httpd.service
```

```
1  [Unit]
2  Description=Apache Web Server
3
4  [Service]
5  Type=forking
6  PIDFile=/run/httpd/httpd.pid
7
8  ExecStart=/usr/bin/apachectl start
9  ExecStop=/usr/bin/apachectl graceful-stop
10 ExecReload=/usr/bin/apachectl graceful
```

# Pages de manuel correspondantes

- Toutes les pages de man sont listées dans systemd.**index(7)** ;
- Toutes les options d'unit sont listées dans systemd.**directives(7)** ;
- Options génériques pour toutes les units dans systemd.**unit(5)** ;
- Options spécifiques aux services dans systemd.**service(5)**.

# Exemple : **systemctl status**

```
$ sudo systemctl status cronie.service
● cronie.service - Periodic Command Scheduler
   Loaded: loaded (/usr/lib/systemd/system/cronie.service; enabled; vendor preset: disabled)
   Active: active (running) since mar. 2016-01-05 03:01:07 CET; 1 weeks 1 days ago
 Main PID: 575 (crond)
    Tasks: 1 (limit: 512)
   CGroup: /system.slice/cronie.service
           └─575 /usr/bin/crond -n

janv. 12 17:48:01 hydra anacron[23364]: Job `cron.weekly' terminated
janv. 12 18:01:01 hydra CROND[23982]: (root) CMD (run-parts /etc/cron.hourly)
janv. 12 19:01:01 hydra CROND[26767]: (root) CMD (run-parts /etc/cron.hourly)
janv. 13 13:01:01 hydra CROND[14366]: (root) CMD (run-parts /etc/cron.hourly)
janv. 13 13:01:01 hydra anacron[14371]: Anacron started on 2016-01-13
janv. 13 13:01:01 hydra anacron[14371]: Will run job `cron.daily' in 10 min.
janv. 13 13:01:01 hydra anacron[14371]: Jobs will be executed sequentially
janv. 13 13:40:53 hydra anacron[14371]: Job `cron.daily' started
janv. 13 14:01:01 hydra CROND[15078]: (root) CMD (run-parts /etc/cron.hourly)
janv. 13 15:01:01 hydra CROND[8012]: (root) CMD (run-parts /etc/cron.hourly)
```

# Gestion des dépendances

- Gestion implicite :
  - Activation via une socket TCP ou UNIX ;
  - Activation via D-Bus.

- Gestion explicite :
  - Dépendance : **Requires=** / **Wants=** :
    - `systemctl show -p Requires nginx`
    - `systemctl show -p Wants nginx`
    - `systemctl list-dependencies nginx`
    - `systemctl list-dependencies --reverse nginx`
  - Ordre : **After=** / **Before=** :
    - `systemctl show -p After nginx`
    - `systemctl show -p Before nginx`
    - `systemctl list-dependencies --after nginx`
    - `systemctl list-dependencies --before nginx`

# Comment ordonner le lancement d'un service ?

Démarrer nginx après mysql et seulement si son lancement a réussi :

```
$ cat /etc/systemd/system/nginx.service.d/after-mysql.conf
```

```
1 [Unit]
2 Requires=mysql.service
3 After=mysql.service
```

# Units : **target**

- Remplaçant des runlevels SysVinit ;
- Points de synchronisation ;
- Regroupe plusieurs units à l'aide des dépendances ;
- Plus flexible (pas de limite en quantité et simultanéité).

# Units : **multi-user.target**

```
[Unit]
Description=Multi-User System
Documentation=man:systemd.special(7)

Requires=basic.target
Conflicts=rescue.service rescue.target
After=basic.target rescue.service rescue.target

AllowIsolate=yes
```

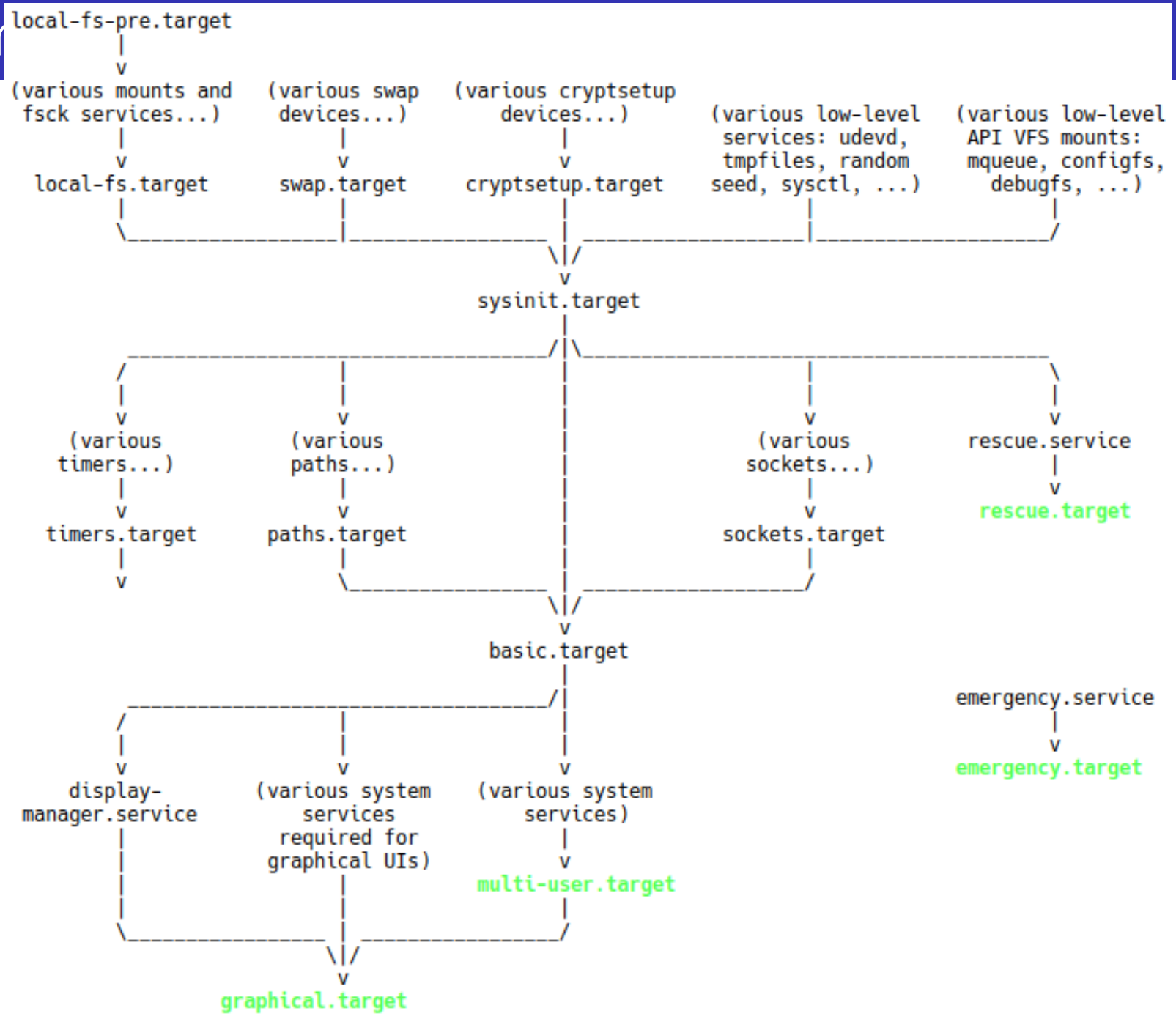
# Units : **httpd.service**

```
1  [Unit]
2  Description=Apache Web Server
3  After=network.target remote-fs.target nss-lookup.target
4
5  [Service]
6  Type=forking
7  PIDFile=/run/httpd/httpd.pid
8
9  ExecStart=/usr/bin/apachectl start
10 ExecStop=/usr/bin/apachectl graceful-stop
11 ExecReload=/usr/bin/apachectl graceful
12
13 [Install]
14 WantedBy=multi-user.target
```

# Démarrage du système

- Vers quelle `target` pointe `default.target` ?
- Boot $\Leftrightarrow$ `systemctl start default.target`
- Résolution des dépendances ;
- Démarrage des units ;
- Apparition des devices.

# Dém


local-fs-pre.target
(various mounts and fsck services...)
(various swap devices...)
(various cryptsetup devices...)
(various low-level services: udevd, tmpfiles, random seed, sysctl, ...)
(various low-level API VFS mounts: mqueue, configfs, debugfs, ...)
local-fs.target
swap.target
cryptsetup.target
sysinit.target
(various timers...)
(various paths...)
(various sockets...)
rescue.service
timers.target
paths.target
sockets.target
rescue.target
basic.target
emergency.service
emergency.target
display-manager.service
(various system services required for graphical UIs)
(various system services)
multi-user.target
graphical.target

# Usage : Démarrage au boot

```
# systemctl enable httpd.service
ln -s /usr/lib/systemd/system/httpd.service \
      /etc/systemd/system/multi-user.target.wants/
```

```
# systemctl disable httpd.service
rm /etc/.../multi-user.target.wants/httpd.service
```

```
# systemctl mask httpd.service
ln -s /dev/null /etc/systemd/system/httpd.service
```

# Usage : Démarrage lorsque le réseau est OK

Exemple sous Debian :

- Target : `network-online.target` ;
- Pas de dépendance par défaut ;
- Choisir ce qui permet d'atteindre l'état « réseau OK » ;

# Usage : Démarrage lorsque le réseau est OK

Exemple sous Debian :

- Target : `network-online.target` ;
- Pas de dépendance par défaut ;
- Choisir ce qui permet d'atteindre l'état « réseau OK » ;
- Les interfaces définies dans le fichier `/etc/network/interfaces` sont configurées :
  - `$ ln -s /lib/systemd/system/networking.service /etc/systemd/system/network-online.target.requires/`

# Usage : Démarrage lorsque le réseau est OK

Exemple sous Debian :

- Target : `network-online.target` ;
- Pas de dépendance par défaut ;
- Choisir ce qui permet d'atteindre l'état « réseau OK » ;
- Les interfaces définies dans le fichier `/etc/network/interfaces` sont configurées :
  - `$ ln -s /lib/systemd/system/networking.service /etc/systemd/system/network-online.target.requires/`
- Service devant attendre que le réseau soit OK :
  - `$ cat /etc/systemd/system/foo.service.d/bar.conf`

```
1 [Unit]
2 Requires=network-online.target
3 After=network-online.target
```

# Interaction **udevd** / **systemd**

- Entrée dans `/etc/fstab` :

```
1 /dev/sda3   /home   ext4   rw,nodev,nosuid   0 2
```

- Exemple :
  0. `systemd-fstab-generator` ⇒ `home.mount`
  1. Attends la découverte du périphérique de type block : `dev-sda3.device` ;
  2. Vérification du système de fichier : `systemd-fsck@dev-sda.service` ;
  3. Monte le système de fichier.

# Débugger **systemd** ?

Un démon ne marche pas ?

- `# systemctl --state failed`
- `# systemctl status foo.service`
- `# systemctl show foo.service`
- `# journalctl -e`
- `# journalctl -e -u foo.service`
- `# journalctl -e /usr/bin/foo`
- `# journalctl -e _PID=123456`

# Débugger **systemd** ?

Erreurs au démarrage ?

- Erreur courante : configuration `/etc/fstab` incorrecte ;
- Options noyau :
  - `systemd.unit=rescue.target`
  - `systemd.unit=emergency.target`
  - `systemd.log_level=debug systemd.log_target=kmsg log_buf_len=1M`
  - `systemd.debug-shell` (`tty9`)
  - `systemd.confirm_spawn=true`
- `# systemctl list-jobs`

# Débugger **systemd** ?

Récupérer les logs des précédents démarrage :

- Dans `/etc/systemd/journald.conf` :

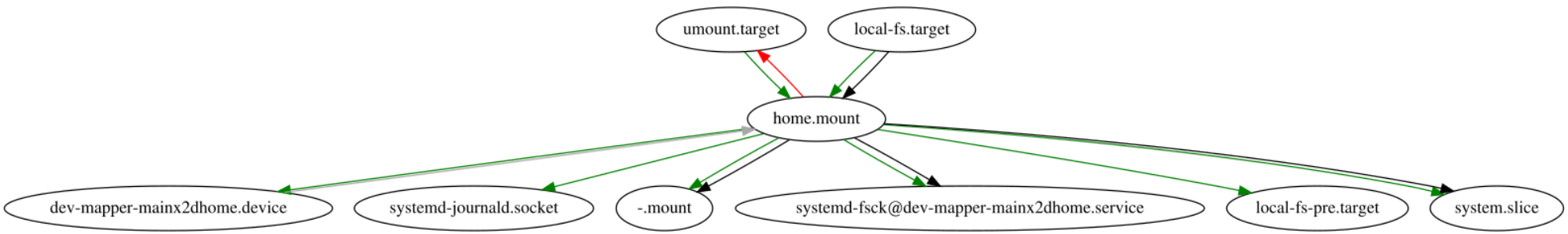
```
1 Storage=persistent
```

- `# journalctl -b -1`

# Débugger **systemd** ?

Problème de dépendance ?

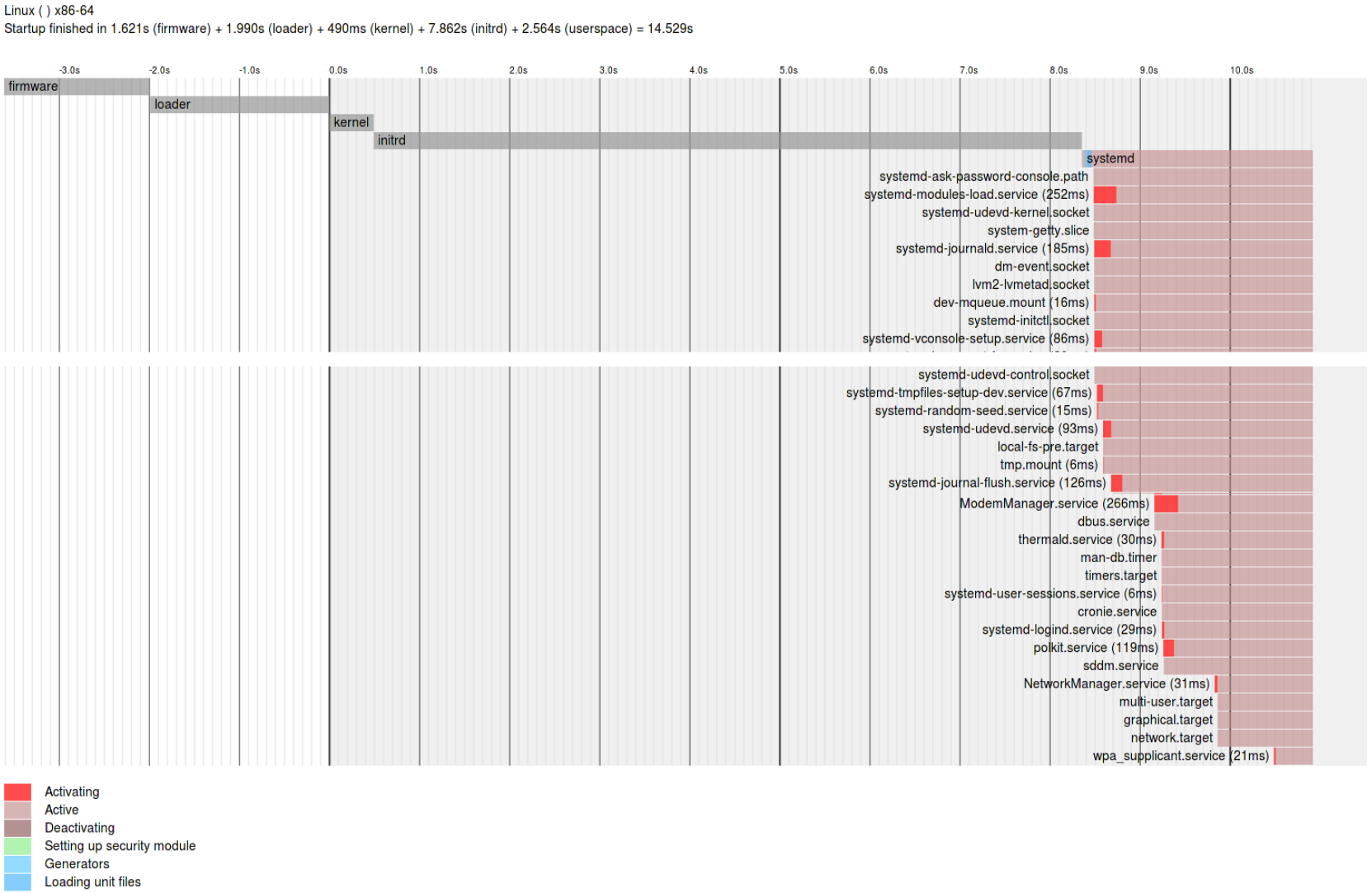
- `# systemctl list-dependencies`
- `# systemd-analyze dot home.mount | dot -Tsvg > plt.svg`
- `$ firefox plt.svg`



# Débugger **systemd** ?

## Problème de performance ?

- `# systemd-analyze plot > boot.svg`



# Débugger **systemd** ?

## Problème de performance ?

```
$ systemd-analyze critical-chain
The time after the unit is active or started is printed after the "@" character.
The time the unit takes to start is printed after the "+" character.

graphical.target @1.505s
└─multi-user.target @1.505s
  └─NetworkManager.service @1.473s +31ms
    └─firewalld.service @800ms +672ms
      └─basic.target @798ms
        └─sockets.target @798ms
          └─sshd.socket @798ms
            └─sysinit.target @797ms
              └─systemd-timesyncd.service @763ms +33ms
                └─systemd-tmpfiles-setup.service @746ms +13ms
                  └─local-fs.target @745ms
                    └─boot.mount @731ms +13ms
                      └─systemd-fsck@dev-sda1.service @681ms +49ms
                        └─dev-sda1.device @680ms
```

# Débugger **systemd** ?

Documentation :

- http://freedesktop.org/wiki/Software/systemd/Debugging/
- https://fedoraproject.org/wiki/How_to_debug_Systemd_problems
- https://wiki.archlinux.org/index.php/Systemd#Troubleshooting
- https://wiki.debian.org/systemd#Debugging

# Design et fonctionnalités : Standardisation

- Standardisation : chaque distribution avait ses propres scripts d'init et ses propres fichiers de configurations.
- Tout est maintenant commun :
  - `/etc/hostname` $\Rightarrow$ `hostnamectl` ;
  - `/etc/locale.conf`, `/etc/vconsole.conf` $\Rightarrow$ `localectl` ;
  - `/etc/localtime` $\Rightarrow$ `timedatectl`.
- Mais pas encore supporté par toutes les distributions (attention notamment sur Debian & Ubuntu).

# /run

- systemd a introduit le dossier `/run` ;
- Utilise le système de fichier `tmpfs` pour stocker les données temporairement ;
- Disponible très tôt au démarrage ;
- Utilisable pour stocker les informations non permanentes liées aux services ;
- Voir **file-hierarchy(7)**.

# tmpfiles.d

- `/{etc,usr/lib}/tmpfiles.d/*.conf`
- Liste de dossiers ou fichiers à créer automatiquement au démarrage ;
- Exemple : `$ cat /usr/lib/tmpfiles.d/tuned.conf`

```
1 d /run/tuned 0755 root root -
```

- Voir **tmpfiles.d(5)**.

# sysusers.d

- `/{etc,usr/lib}/sysusers.d/*.conf`
- Liste d'utilisateur système à créer automatiquement au démarrage si ils n'existent pas déjà ;
- Exemple : `$ cat /usr/lib/sysuers.d/mariadb.conf`

```
1 u mysql - "MariaDB" /var/lib/mysql
```

- Voir **sysusers.d(5)**.

# Plan


3 systemd

- Historique
- Fonctionnement
- journald
- logind, networkd, timesyncd, resolved, etc.
- Sécurité

# Design

- Similaire à syslog(-ng)/rsyslog : Récupère les messages envoyés parles processus sur la socket /dev/log et les stocke proprement ;
- Démon lancé très tôt au boot et fortement intégré avec systemd ;
- Cette intégration permets d'obtenir beaucoup plus d'informations que ce qui normalement possible avec (r)syslog(-ng) ;
- Utilise un format de stockage personnalisé et compressé [16] ;

# Commandes usuelles

- `journalctl -e` : affiche les logs les plus récents du boot courant ;
- `journalctl -b -1` : affiche les logs du boot précédent ;
- `journalctl -b -0 _PID=333` : limite l'affichage aux logs émis par le PID 333 pour ce boot ;
- `journalctl -k` : équivalent de dmesg ;
- `journalctl -u nginx(.service)` : affiches tous les logs liésau service nginx ;
- `journalctl -f -n 50` : affiche les 50 derniers logs puis affiche les logs au fur et à mesure ;
- `journalctl /usr/bin/dbus-daemon` : tous les logs émis par le binaire `/usr/bin/dbus-daemon`.

# Plan


3 systemd

- Historique
- Fonctionnement
- journald
- logind, networkd, timesyncd, resolved, etc.
- Sécurité

# Autres projets rattachés

- logind : gère les sessions utilisateurs (remplace ConsoleKit) ;
- networkd : gestion minimale « statique » du réseau pour ne plus dépendre de scripts à ce niveau ;
- timesyncd : client NTP minimal (client uniquement) ;
- resolved : resolveur DNS ;
- etc.

# Plan


3 systemd

- Historique
- Fonctionnement
- journald
- logind, networkd, timesyncd, resolved, etc.
- Sécurité

# Fonctionnalité de sécurité

- Support de toutes les options de gestion de processus proposées par le noyau :
  - UID, GID ;
  - limites & Co.
  - cgroups v1 & v2 ;
  - namespaces ;
  - capabilities ;
  - seccomp ;
  - etc.
- Exemples détaillés avec mise en place dans la présentation : Durcissement système à l'aide de systemd (SSTIC 2017) [17].

# Sécurité de systemd ?

- Quelques vulnérabilités majeures : CVE-2018-15686, CVE-2018-15688,CVE-2016-7795, CVE-2016-10156, CVE-2013-4392, CVE-2013-4327,CVE-2012-1174, CVE-2012-0871 ;
- Bugs médiatiques : CVE-2017-1000082, CVE-2017-9445, CVE-2017-15908 ;
- Écrit en C mais pratiquement toutes les erreurs sont logiques (non liées au langage) ;
- Surface d'attaque très limitée ;
- Démons systèmes correctement confinés (voir vulnérabilités resolved).

# Plan


1 Infrastructures

2 Conteneurs

3 systemd

4 Gestionaires de conteneurs

- Implémentations
- Standards ?
- Autres usages

5 SELinux

6 Conteneurs et sécurité

7 Virtualisation et sécurité

8 Sécurité des infrastructures

# Plan

4 Gestionaires de conteneurs

- Implémentations
- Standards ?
- Autres usages

# Création d’un conteneur sous Linux

Éléments nécessaires :

- gestion des ressources et de l’accès à certains périphériques avec les cgroups ;
- isolation (modulable) avec les namespaces ;
- restrictions de permissions avec les capabilities, les filtres seccomp et les Linux Security Modules (SELinux / AppArmor) ;
- image du conteneur (arborescence propre au conteneur : contenu du /) ;
- accès au réseau.

# Container runtime & Container engine

## *Container runtime*

Programme responsable de la création d'un conteneur et de la configuration de l'environnement d'exécution :

- mise en place des cgroups & limites UNIX ;
- mise en place des namespaces ;
- configuration des capabilities, filtres seccomp, LSM.

## *Container engine*

Programme responsable de la gestion du cycle de vie d'un conteneur :

- gestion des images de conteneurs (création, récupération à distance, envoi, instanciation, etc.) ;
- gestion des points de montage, volumes persistants, etc.
- mise en place des interfaces réseau (virtuelles) ;
- suivit des conteneurs en cours d'exécution.

Utilise un container runtime pour lancer les conteneurs.

**Attention :** Nomenclature et limite floue, souvent mélangé !

# Implémentations sous Linux

- LXC/LXD (2008) : LinuX Containers initialement développé par IBM. Maintenu par Ubuntu (Canonical) ;
- systemd-nspawn (2010) : Développé comme outils de test et remplaçant de chroot. Base de rkt. Maintenu ;
- Docker/Moby (2013) : Rends populaire les concepts de partage, copie, redistribution d'image de conteneurs. Maintenu par Docker Inc ;
- rkt (2014) : Créé par CoreOS pour corriger certains défauts et limitations de Docker. Futur incertain ;
- runc et containerd (2015) : Séparation de Docker en plusieurs projets indépendants ;
- Clear Containers (2015) : Créé par Intel, combine les conteneurs et la virtualisation matérielle avec KVM. Désormais regroupé avec runV dans le projet Kata Containers ;

# Implémentation sous Linux

- Minijail (2016) : Créé par Google pour ChromeOS & Android.
- railcar (2017) : Ré-implémentation de runc en Rust ;
- CRI-O (2017) : Ré-implémentation d'une partie de Docker pour usage avec Kubernetes uniquement. Maintenu par Red Hat ;
- Podman (2018), Buildah & Skopeo : Ré-implémentation des fonctionnalités de Docker sans démon. Maintenu par Red Hat ;
- gVisor (2018) : Isolation avec un noyau écrit en Go. Maintenu par Google ;
- Nabla containers (2018) : Isolation à l'aide d'un unikernel. Maintenu par IBM ;

# LXC/LXD

- Objectif initial : faire tourner un système complet dans un conteneur ;
- Image : simple `tar.gz` d'un `/`, obtenu avec `debootstrap`, `dnf -install-root=...` ou équivalent ;
- Création de conteneurs à partir de templates ;
- Avantages :
  - Terrain connu, fonctionnement prévisible ;
  - Aussi flexible qu'un système classique.
- Désavantages :
  - Duplication des services (journalisation, monitoring...) ;
  - Consommation en ressources (mémoire et disque) ;
  - Plus de gestion nécessaire (plus proche de la virtualisation « classique ») : mises à jour, sauvegarde...
- LXD : gestionnaire de conteneur LXC.

# Docker/Moby

- Objectif : faire tourner une seule application par conteneur ;
- Dockerfile : sorte de script pour créer des conteneurs ;
- DockerHub/Registry : partage d'images de conteneurs ;
- Avantages :
  - Simplificité (récupération des images, utilisation) ;
  - Consommation en ressources (mémoire et dique) réduite ;
  - Gestion à l'échelle d'une application et non plus d'un système.
- Désavantages :
  - Chaque conteneur doit être maintenu à jour indépendamment ;
  - Les applications classiques ne sont pas prévues pour fonctionner en tant que PID 1 dans un conteneur : processus zombie, etc.
  - Aucun service support n'est disponible dans le conteneur.

# Podman, Buildah, Skopeo

- Ré-écriture de Docker sans utiliser de démon central ;
- Buildah : Construction d'image de conteneur à partir d'un Dockerfile ou à l'aide de commande arbitraires ;
- Skopeo : Récupération d'images distante stockée dans un registre et envoi d'images locale ;
- Avantages :
  - Plus de problème liés au démon Docker : conteneur dans un conteneurs, redémarrage, etc.
  - Bonne intégration avec systemd ;
  - Implémentation de fonctionnalités très demandées mais jamais ajoutées à Docker ;
  - Séparation des tâches $\Rightarrow$ meilleure sécurité.
- Désavantages :
  - N'implémente pas encore toutes les fonctionnalités de Docker ;
  - Moins « populaire » que LXC ou Docker.

# Plan

4 Gestionaires de conteneurs

- Implémentations
- Standards ?
- Autres usages

# Standards ?

- La prolifération des outils de gestion de conteneurs a poussé à la standardisation [92] ;
- Création de l'Open Container Initiative et de la Cloud Native Computing Foundation ;
- De facto standard : Docker Format v2 ;
- OCI Image Format Specification ;
- OCI Runtime Specification.

# Plan

4 Gestionaires de conteneurs
- Implémentations
- Standards ?
- Autres usages

# Conteneurs et applications graphiques

- Sandboxing d'applications graphiques :
  - Firejail ;
  - Flatpak (& Flathub) ;
  - Snapcraft (& Snaps).
- Attention : Wayland nécesssaire. Le serveur d'affichage X.org ne peut pas être configuré de façon sécurisé.

- A l'échelle d'un système d'exploitation :
  - Fedora Silverblue ;
  - Endless OS ;
  - Subgraph OS.

# Plan


1 Infrastructures

2 Conteneurs

3 systemd

4 Gestionaires de conteneurs

5 SELinux

- Concepts fondateurs
- Modèles de contrôle d'accès
- SELinux
- SELinux en pratique

6 Conteneurs et sécurité

7 Virtualisation et sécurité

8 Sécurité des infrastructures

# Plan

5 SELinux

- Concepts fondateurs
- Modèles de contrôle d’accès
- SELinux
- SELinux en pratique

# La triade DIC (CIA triad)

- Les fondamentaux :
  - Confidentialité : qui a accès à une information ?
  - Intégrité : qui peut écrire, modifier ou créer de l'information ?
  - Disponibilité : est-ce qu'une ressource est disponible ?
- Les extensions :
  - Authenticité : comment établir une identité ?
  - Auditabilité : comment tracer les opérations effectuées ?
  - Non répudiation : est-il possible de nier avoir fait une action ?

# Le principe du moindre privilège

> « Tous les programmes et utilisateurs d'un système doivent fonctionner en utilisant le minimum de privilèges nécessaires pour mener à bien leurs fonctions. »
>
> — Jerome Saltzer, *Communications of the ACM*

Pour mettre en place un tel modèle d'opération, il faut :

- séparer les privilèges ;
- minimiser les privilèges ;
- confiner ou cloisonner les programmes privilégiés.

# Plan

5 SELinux

- Concepts fondateurs
- Modèles de contrôle d’accès
- SELinux
- SELinux en pratique

# DAC : Contrôle d'Accès Discrétionnaire

- Le DAC est le modèle utilisé par défaut sous Linux ;
- Chaque utilisateur a le contrôle des fichiers et programmes lui appartenant :
  - création / modification / suppression
- Les programmes lancés disposent des même droits que l'utilisateur
- Les utilisateurs peuvent donner des droits sur leur ressources à leur *discrétion* :
  - chmod / chown
- Le niveau de sécurité dépend de la bonne séparation entre utilisateurs et du niveau de sécurité de chaque application.

# Les limites du DAC

- Il est possible d'utiliser les *capabilities* et différents utilisateurs pour séparer les privilèges accordés aux applications ;
- Mais cette séparation n'est pas toujours assez fine :
  - de nombreuses *capabilities* sont équivalente à `root` (cf. section sur les *capabilities*) ;
- Il n'est pas possible de séparer des programmes s'exécutant sous un même identifiant UNIX.

Si programme s'exécutant sous l'identité `root` est compromis, l'attaquant obtient les droits `root` et l'ensemble du système est compromis.

# Contrôle d’accès obligatoire ou Mandatory Access Control (MAC)

- Les droits d’accès aux objets du système ne sont plus à la discrétion des utilisateurs ;
- L’administrateur du système définit une politique de sécurité qui décrit quel ressource est accessible à quel utilisateur.

# Contrôle d’accès obligatoire sous Linux

- Intervient après le contrôle d’accès discrétionnaire standard ;
- Implémenté sous forme de *Linux Security Module (LSM)* ;
- Implémentations disponibles sous Linux :
  - SELinux (utilisé par défaut sur Fedora, CentOS, RHEL) ;
  - AppArmor (utilisé par défaut sur Ubuntu, Debian) ;
  - SMACK (utilisé par défaut sur Tizen) ;
  - Tomoyo ;
  - grsecurity RBAC.

# Plan


5 SELinux

- Concepts fondateurs
- Modèles de contrôle d'accès
- SELinux
- SELinux en pratique

# SELinux ?

- Security-Enhanced Linux ;
- Implémentation d'un contrôle d'accès obligatoire ;
- Développé initiallement par la NSA ;
- Premier MAC intégré dans le noyau Linux (2.6+) ;
- Implémentation du contrôle d'accès dans le noyau ;
- Outils en espace utilisateur pour manipuler la politique.

# SELinux & Android

- Utilisé par Android depuis la version 4.4 (KitKat), ~96% des *devices* ;
- Protection complète depuis la version 5.0 (Lollipop), ~89% des *devices* ;
- Impact significatif sur le durcissement du système [81,80].

# Concepts

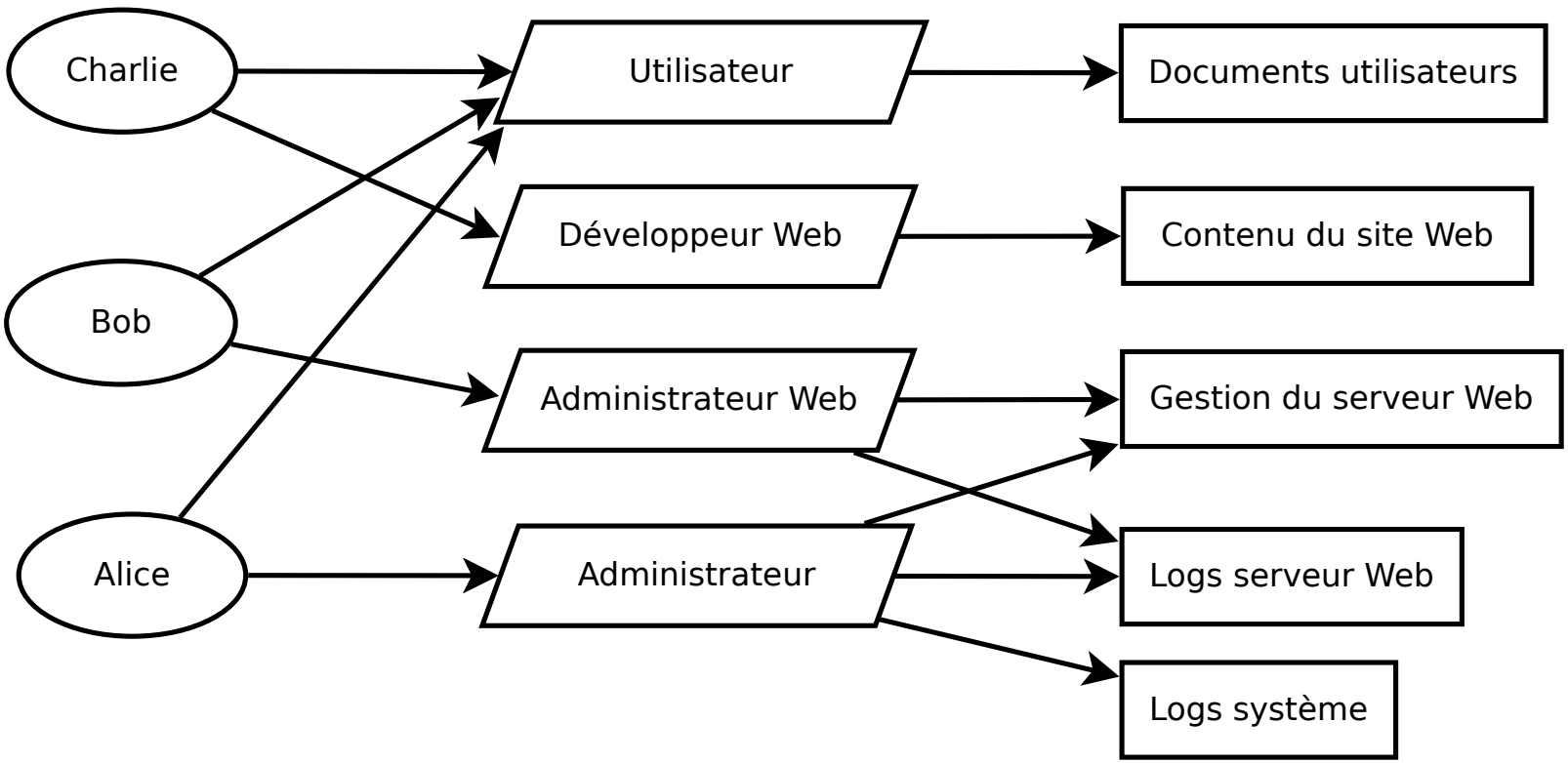
- Basé sur l'architecture *Flask* :
  - Moniteur de référence dans le noyau ;
  - Stocke la politique de contrôle d'accès ;
  - Prends les décision de contrôle d'accès au niveau des appels systèmes (contrôle à *grain fin*).

# Politique de sécurité

- Décription exhaustive de l'ensemble des opérations autorisées sur un système ;
- Pour cela, on distingue :
  - Les sujets : entités actives du système (processus) ;
  - Les objets : entitiés passives du système (fichier, socket, etc.).
- La politique décrit quelles actions les sujets peuvent faire sur les objets ou sujets du système.

# Role Based Access Control (RBAC)

- Sous concept pour mettre en place le MAC ;
- Objectif : factoriser les opérations réalisables sur des objets en un rôle ;
- Association des utilisateurs à un ou plusieurs rôles.



# Politique

- Déclare les identiés, rôles et types ;
- Définition des règles d'autorisation d'accès ;
- Définition des règles de transitions de types ;
- Définition des contraintes, etc.

- Initialement écrit dans le Policy Language ;
- Vérifications à la compilation des contraintes ;
- Utilisation de macros/fonctions ;
- Remplaçant : SELinux Common Intermediate Language (CIL) [83] ;
- Même concepts, syntax différente.

# Type Enforcement (TE)

- SELinux associe à chaque ressource du système un contexte de sécurité (Security Context ou SC) ;
- Lorsqu'ils sont associés à des objets du système on les appelle aussi *label* ;
- Les décisions de contrôle d'accès prises par le noyau prennent en compte ces SC ;
- Dans le cas de SELinux, ce contrôle est appelé *Type Enforcement*.

# Intéraction

Chaque régle SELinux autorisant une intéraction est constituée :

- d'un contexte de sécurirté source (processus)
- qui effectue une opération :
  - class : file dir, process, socket (type de l'opération) ;
  - permission : read, write, unkink, accept (action) ;
- sur un contexte de sécurirté destination (fichier, socket, processus, etc.).

# Contexte de sécurité

- Ensemble d'attributs sous forme de texte :
  - `system_u:system_r:httpd_t` : contexte du processus apache ;
  - `user_u:object_r:user_home_t` : contexte de fichier d'un répertoire utilisateur.
- Trois attibuts principaux : `user:role:type`
  - User : identité SELinux ( != DAC) propriétaire du contexte ;
  - Role : modèle RBAC
    - rôle courant de l'identité pour un sujet ;
    - rôle par défaut pour les objets (`object_r`).
  - Type : modèle TE
    - Domaine dans lequel s'exécute un sujet ;
    - Type associé à un objet.

# Contexte de sécurité

- Pour les processus :
  - attribué à la création ;
  - ne peut évoluer que lors d'un appel à `execve`.

- Pour les fichiers :
  - stocké dans les attributs étendus du fichier (*xattr*) ;
  - peut être modifié à volonté.

# Type

- Un type est un identifiant ;
- Pas de signification intrinsèque : il est contraint aux règles de la politique ;
- Regroupe les sujets et objets ayant les même autorisations.

- On parle de :
  - type pour un objet ;
  - domaine pour un sujet.

- Exemples : *system_t*, *user_t*, *unconfined_t*

# Convention de nommage

- applicationd_t : domaine associé à un démon ;
- application_t : domaine associé à un processus classique ;
- application_exec_t : fichier binaire de l'application ;
- application_conf_t : fichier de configuration ;
- application_log_t : fichier de log ;
- application_file_t : fichiers divers ;
- application_lib_t : bibliothèque ;
- application_tmp_t : fichier temporaire ;
- applicationd_unit_file_t : fichier d'unit systemd.

Un objet/processus ne possède qu'un seul type mais un type peut être associé à plusieurs objets.

# Types

Chaque type doit être déclaré avec utilsation :

```
# sujet
type init_t;
type sshd_t;

# objet
type sshd_exec_t
type admin_paswd_exec_t;
```

# Attributs

Attributs : regroupe plusieurs types pour factoriser la politique :

```
# déclaration d'un attribut
attribute file_type;
attribute application_domain_type;
attribute application_exec_type;

# association d'un type (déjà déclaré) à un attribut (déjà
    ↪ déclaré)
typeattribute shadow_t security_file_type;

# déclaration d'un type er association à un attribut (déjà
    ↪ déclaré)
type sshd_exec_t executable_file_type;
```

# Opération

- Opération = droit d'un contexte sujet sur un autre contexte ;
- Permission autorisée sur une classe d'objet ;
- Exemple :
  - file : { open read execute } ;
  - socket : { bind listen }.

# Règle de contrôle d'accès

```
allow <type_source> <type_cible>:<class> { <permissions> };
```

```
# Les sujets avec le type sysadm_t peuvent exécuter (...)
    ↪ les fichiers avec le type ssh_exec_t
allow sysadm_t ssh_exec_t:file {getattr read execute };

# Les sujets avec le type sysadm_t peuvent exécuter (...)
    ↪ les fichiers dont le type a pour attribut
    ↪ application_exec_type
allow sysadm_t application_exec_type:file {getattr read
    ↪ execute ioctl lock execute_no_trans };
```

# Factorisation des permissions

- Macros pour simplifier l'utilisation des permissions : `policy/support/*perms_set.spt`
- Exemples :
  - r_file_perms : permissions pour lire un fichier
    - open, getattr, read, lock, ioctl
  - r_dir_perms : permissions pour traverser un répertoire
    - open getattr read lock search ioctl
  - x_file_perms : permissions pour exécuter un fichier
    - open getattr execute

# Type & file transitions

- Par défaut, le contexte de sécurité d'un nouveau processus est celui de son père ;
- Possibilité de changer de contexte avec une *transition* lors d'un appel à `execve` ;
- Permet de confiner chaque service dans son domaine respectif.

- Par défaut, le contexte de sécurité d'un fichier est celui de son répertoire parent.
- Possibilité d'ajouter des règles pour associer des contextes différents en fonction des répertoire ou noms de fichiers.

# Transition de type : sujet

```
# Lorsque le type user_t crée exécute le fichier ayant le type ssh_exec_t il transite automatiquement vers user_ssh_t
type_transition user_t ssh_exec_t:process user_ssh_t;

# Lorsque le type user_t crée exécute le fichier ayant le type passwd_exec_t il transite automatiquement vers passwd_t
type_transition user_t passwd_exec_t:process passwd_t;

# Règles supplémentaires correspondantes
allow user_t ssh_exec_t : file {read getattr execute}
allow user_ssh_t ssh_exec_t : file entrypoint ;
allow user_r user_ssh_t : process transition ;
```

# Transition de type : objets

```
1  # Lorsque le processus labellé par user_t accède à un
       ↪ fichier (...) de type tmp_t, le type de cet objet
       ↪ transite automatiquement vers user_tmp_t
2  type_transition user_t tmp_t:{ dir file lnk_file sock_file
       ↪ fifo_file } user_tmp_t;
3
4  # Lorsque le processus labellé par sshd_t accède à un
       ↪ fichier (...) de type tmp_t, le type de cet objet
       ↪ transite automatiquement vers sshd_tmp_t
5  type_transition sshd_t tmp_t:{ dir file sock_file }
       ↪ sshd_tmp_t;
6
7  # Règles supplémentaires correspondantes
8  allow user_t tmp_t: dir { open getattr create write };
9  allow user_t user_tmp_t:file rw_file_perms;
10
11 # Lorsque qu'un processus user_t créé un dossier nommé .ssh
       ↪ dans un dossier user_home_dir_t, il est labellé
       ↪ ssh_home_t
12 type_transition user_t user_home_dir_t : dir ssh_home_t
       ↪ .ssh;
```

# Politique Multi-Niveau (MLS & MCS)

Deux attributs supplémentaires ajoutés aux contextes de sécurité :

`unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023`

- MLS : niveaux de sécurité
  - Ajout de contrainte entre différents niveau de sécurité ;
  - Souvent utilisé pour implémenter des modèles de confidentialité.
  - N'est pas utilisé dans la politique *targeted*.
- MCS : catégories
  - Isole plusieurs instances d'un même type ;
  - Le '.' définie un ensemble (ex : c0.c16) ;
  - La ',' définie une liste (ex : c21,c36,c45) ;
  - Les deux séparateurs peuvent être combinés (ex : c0.c16,c21,c36,c45).

# Politique Multi-Niveau (MLS & MCS)

- Contrôle entre les niveaux et les catégories est réalisé via des contraintes SELinux ;
- Décrites dans la politique : `policy/mls`, `policy/mcs` ;
- Tous les types ne sont pas nécessairement constraints ;
- Utilisé principalement pour confiner les machines virtuelles et les conteneurs (cf. chapitre sécurité & conteneurs, sécurité & KVM).

# Plan

5 SELinux

- Concepts fondateurs
- Modèles de contrôle d'accès
- SELinux
- SELinux en pratique

# Refpolicy

- Strict policy ;
- Tous les utilisateurs sont confinés ;
- La politique doit décrire l'intégralité des intéractions autorisées ;
- Disponible principalement sous Gentoo Hardened (Debian & Arch Linux) ;
- Restrictif et complexe.

# Fedora, CentOS, Red Hat Enterprise Linux

- Targeted policy ;
- Les démons système sont confinés ;
- Les utilisateurs intératifs ne sont pas confinés par défaut (`unconfined`) ;
- Possibilité de confiner certains utilisateurs.

# Politique et état de SELinux

- Configuration de SELinux et politique actuelle stockées dans `/etc/selinux` ;
- `sestatus -v` : état général ;
- État actuel de SELinux dans le noyau : `/sys/fs/selinux/` ;
- État actuel de la politique stocké dans `/var/lib/selinux/` ;
- `seinfo -v` : informations sur la politique.

# Audit

- SELinux utilise le sous système d'audit du noyau ;
- Démon `auditd` en espace utilisateur ;
- Logs dans `/var/log/audit/audit.log` ou dans le Journal.

- Loggue par défaut tous les accès refusés ;
- Outils `audit2allow` et `audit2why` pour générer des règles de politique à partir de ces logs.

# Exemple de messages AVC

```
1 type=AVC msg=audit(1515257682.718:409): avc:  denied  {
    ↪ getattr } for  pid=4320 comm="httpd"
    ↪ path="/home/fedora/public_html/test" dev="sda1"
    ↪ ino=262285 scontext=system_u:system_r:httpd_t:s0
    ↪ tcontext=unconfined_u:object_r:httpd_user_content_t:s0
    ↪ tclass=file permissive=0
2 type=AVC msg=audit(1515257956.228:527): avc:  denied  {
    ↪ setuid } for  pid=4646 comm="sudo" capability=7
    ↪ scontext=user_u:user_r:user_t:s0
    ↪ tcontext=user_u:user_r:user_t:s0 tclass=capability
    ↪ permissive=0
```

# Règle de contrôle d'accès : audit

```
1  # Audite l'accès en lecture au type shadow_t par sysadm_t
2  auditallow sysadm_t file:read shadow_t;
3
4  # Les types non privilégiés ne peuvent pas lire shadow_t
5  # Utilisation de ~ pour la négation
6  neverallow ~can_read_shadow_passwords shadow_t:file read;
7
8  # Ne pas auditer certains accès aux répertoires pour le
       ↪ type staff_t
9  # Utilisation de - pour retirer le type security_file_type
       ↪ de file_type
10 dontaudit staff_t { file_type -security_file_type }: dir {
       ↪ getattr search read lock};
```

# Politique modulaire

- Politique source stockée sous forme d'une collection de fichiers texte ;
- Politique séparée entre une base et de nombreux modules ;
- « Compilation » de la base et des modules pour former la politique finale, sous forme binaire.
- Seule la forme finale est chargée par le noyau.

- On écrit rarement une politique de zéro ;
- On ajoute ou modifie un module existant.

# Module SELinux

- Composé de trois fichiers :
  - `.te` : ensemble des règles SELinux à appliquer ;
  - `.fc` : list des associations entre chemin de fichiers et contexte (*fc : file contexts*) ;
  - `.if` : interface, ou ensemble de « fonctions » proposé pour utilisation dans d'autres modules.
- Création d'un module à l'aide de `/usr/share/selinux/devel/Makefile` ;
- Installation avec `semodule -install -priority=500`.

# Exemple de module SELinux

toto.te :

```
policy_module(toto, 0.0.1)

require {
  type httpd_t;
  type httpd_sys_script_t;
  type user_tmp_t;
}

type toto_data_t;
files_type(toto_data_t);

allow httpd_t toto_data_t:file read_file_perms;
allow httpd_t toto_data_t:dir search_dir_perms;

allow httpd_sys_script_t user_tmp_t:file read_file_perms;
```

# Exemple de module SELinux

toto.fc :

```
1 /var/www/html/toto                -d
      ↪ gen_context(system_u:object_r:toto_data_t,s0)
2 /var/www/html/toto(/.*)?          --
      ↪ gen_context(system_u:object_r:toto_data_t,s0)
```

# Ecrire un module SELinux

1. `sepolgen` : génère un caneva à partir de template pour les modules couramment ajoutés.
2. « Compilation » du module ;
3. Chargement du module ;
4. Tests ;
5. Lecture des logs ;
6. Modification du module ;
7. Goto 2.

Fin : Désactivation du mode permissif.

# Contextes de sécurité

- Pour les processus :
  - `id -Z;`
  - `ps -Z.`
- Pour les fichiers :
  - `ls -Z;`
  - `mkdir -Z;`
  - `cp -Z;`
  - `mv -Z;`
  - `find -context.`

# Gestion des contextes

- `restorecon` : restaure le contexte associé à un fichier par la politique :
  - -R : récursif ;
  - -v : verbeux ;
  - -F : restaure aussi le rôle et l'utilisateur SELinux.
- `chcon` : modification arbitraire du contexte d'un fichier.
  - -u, -r, -t : défini l'utilisateur, rôle, type ;
  - -R : récursif ;
  - –reference=file : utilise le contexte d'un autre fichier comme référence.
- `fixfiles onboot` : restore les contextes associés à tous les fichiers du système au prochain boot.

# Points de montage

- Options à passer à `mount` :
- Force les contexte pour tous les fichiers : `-o context=user:role:type`
- Contexte par défaut pour les fichiers sans contexte : `-o defcontext=user:role:type`
- Contexte pour le système de fichier : `-o fscontext=user:role:type`

# Manipulation de la politique

- `semanage` : gestion de la politique SELinux
  - login : association entre les utilisateurs Linux et SELinux ;
  - user : utilisateurs SELinux
  - port : association contextes et ports réseau
  - module : modules de la politique
  - fcontext : associations contextes et fichiers
  - boolean : gestion des booléens ;
  - dontaudit : active/désactive les règles *dontaudit*.
  - etc.
- `semodule -DB` : désactiver les règles `dontaudit`.

# Booléens

- Une fois chargée, la politique est immuable ;
- Possibilité d'ajouter des conditions pour autoriser dynamiquement certaines opérations sans avoir à changer la politique ;
- Commandes `getsebool` & `setsebool` ou `semanage boolean`.

# Recherches dans la politique

- `sesearch` : recherche de règles dans la politique.

- Liste tous les fichiers que le contexte `httpd_t` peut lire :
- sesearch -v –allow -s httpd_t -c file -p read

- Liste tous les contextes pouvant écrire dans les fichiers `shadow_t` :
- sesearch -v –allow -t shadow_t -c file -p write

- Liste toutes les règles associées au booléen `samba_enable_home_dirs` :
- sesearch -v –allow -b samba_enable_home_dirs

# Inspection de la politique

- `sepolicy` :
  - booleans : informations sur les booléens ;
  - communicate : est-ce que deux domaine peuvent intéragir entre eux ?
  - generate : générer des modules à partir de template ;
  - gui : lance les outils graphiques ;
  - interface ;
  - manpage : générer les pages de man de SELinux ;
  - network : relation entre ports et types SELinux ;
  - transition : est-ce qu'un domaine peut transiter vers un autre ?

# Désactiver SELinux ?

- Pour désactiver complètement SELinux, il faut redémarrer le système ;
- `selinux=0` sur la ligne de commande du noyau ou configuration dans `/etc/selinux/config`.
- Plus de MAC, fonctionnement classique DAC.
- Avant de réactiver SELinux, il faut relabéliser intégralement le système de fichiers (`fixfiles onboot`) et redémarrer.

# Désactiver SELinux ?

- `setenforce 0;`
- Mode *Permissive*, qui n'applique plus les contrôle mais loggue uniquement les accés refusés ;
- Le mode permissif ne doit pas être activé sur un système entier en production ;
- Si un démon ne fonctionne pas correctement avec SELinux, il faut passer uniquement ce démon en permissif :
  - semanage permissive -l
  - semanage permissive -a myapp_t
  - semanage permissive -d myapp_t

# Plan

1 Infrastructures

2 Conteneurs

3 systemd

4 Gestionaires de conteneurs

5 SELinux

6 Conteneurs et sécurité

- Sécurité de l'hôte
- Sécurité de l'image
- Sécurité de l'application

7 Virtualisation et sécurité

8 Sécurité des infrastructures

# Points d’attention

- Sécurité de l’hôte hébergeant les conteneurs :
  - Matériel ;
  - Noyau ;
  - OS ;
  - Gestionnaire de conteneurs ;
  - Isolation entre les conteneurs.
- Sécurité de l’image du conteneur :
  - Distribution et intégrité ;
  - Distribution des secrets.
- Sécurité de l’application dans le conteneur :
  - Langage de programmation & Framework ;
  - Système de base ;
  - Mise à jour.

# Plan

6 Conteneurs et sécurité

- Sécurité de l’hôte
- Sécurité de l’image
- Sécurité de l’application

# Matériel

Matériel sécurisé ?

- Intel Management Engine ;
- AMD Platform Security Process ;
- Attacking SMM Memory via Intel CPU Cache Poisoning [62].

# Matériel

Attaques par canaux cachés :

- Pas d'isolation physique : matériel partagé ;
- Attaques utilisant des canaux cachés (ou détournées) [45], [46] :
  - temps & cache : Meltdown & Spectre [60] ;
  - son (GnuPG [47]) ;
  - bruit électromagnétique
  - ...
- Concerne particulièrement les opérations cryptographiques.

# Matériel

Se reposer sur du matériel de confiance :

- TPM : What Trusted Computing Means to Users of CoreOS and Beyond [91].

# Noyau Linux

- Point commun entre tous les conteneurs ;
- Faille exploitable à distance : souvent réseau, très rare ;
- Faille locale : appels systèmes, surface d'attaque large :
  - Gestion de la mémoire (Dirt COW [20]) ;
  - Systèmes de fichiers (overlayfs : [10], partage de `/proc` et `/sys` [11]) ;
  - Drivers : ioctl (cf. Android), . . .
  - Réseau : netfilter. . .
- Retirer des appels systèmes (dur) ;
- Limiter les appels systèmes disponibles : seccomp.

# Noyau Linux : namespaces

- L'isolation fournie par les namespaces et les cgroups a encore des limites ;
- Certaines parties du noyau n'ont pas encore été adaptées aux namespaces ;
- Beaucoup de vulnérabilités trouvées ces dernières années : CVE-2013-1858, [4], [5],[6], [7], [8], [9] ;
- Noyau : cible privilégiée car commun à tous les namespaces ;
- Présentations sur l'exploitation du noyau : [18] et [19].
- Cas non prévus lorsque l'on combine plusieurs namespaces ;
- Failles noyau donc très souvent critiques (« local root ») ;
- User namespace particulièrement touchés.

# Noyau Linux : durcissement

- Initialement : PaX & grsecurity ;
- Aucun patch public et maintenu disponible.
- Travail en cours par le *Kernel Self Protection Project* pour intégrer ces améliorations upstream.

# Système d’exploitation

- Mises à jour ;
- Durcissement standard :
  - compilation durcie (stack protector, RELRO, etc.) ;
  - recommandations génériques [70,73].

# Outils de gestion de conteneurs

- Tous les outils d'infrastructure pour gérer les conteneurs augmentent la surface d'attaque.

# Vulnérabilités spécifiques à LXC

- Vulnérabilité à la création d'un conteneur [30] ;
- Vulnérabilité dans les templates de création de conteneurs [31].

# Vulnérabilités spécifiques à Docker

## Sécurité ?

« Docker is about running random code downloaded from the Internet and running it as root. »

# Vulnérabilités spécifiques à Docker

- Démon `docker` tournant en `root` ;
- Récupération d'images depuis le réseau [35] ;
- Opération sur des images qui ne sont pas de confiance [32] ;
- Beaucoup d'images proposés sur le Registry ne sont pas à jour [36] ;
- Signatures des images pas avant la version 1.8 [33], [34] ;
- Plusieurs vulnérabilités : [37], [38], [39] ;
- Nouvelle architecture depuis la version 1.11.0.

# Spécificités Docker

- Accès au démon docker $\Leftrightarrow$ `root` ;
- Restreindre impérativement l'accès au démon docker : [86] ;
- Plusieurs guides pour utiliser Docker correctement : [84], [85].
- Ne concerne pas Podman (pas de démon).

# Isolation : SELinux

- Objectif : contenir les conteneurs ;
- Utilise les catégories (MCS) ;
- Tous les conteneurs utilisent le même type (même accès) ;
- Associe à l'exécution un ensemble de catégories à chaque conteneur ;
- SELinux garantie qu'il ne peut pas y avoir d'intéraction entre les conteneurs si leur ensemble de catégories sont distincts ;
- Supporté par Docker, rkt et Podman [78], [77], [79] ;
- Limite la portée d'une grande partie des vulnérabilités : ex. CVE-2016-9962 ;
- AppArmor supporté par Docker et Podman.

# Plan


6 Conteneurs et sécurité

- Sécurité de l'hôte
- Sécurité de l'image
- Sécurité de l'application

# Sécurité de l'image

- Restriction des permissions :
  - Utiliser un utilisateur non privilégié ;
  - Limiter les *capabilities* ;
  - Appliquer un filtre *seccomp* strict.
- Image du conteneur en lecture seule (RO) ;
- Stockage persistent externe (par ex. volumes Docker) pour :
  - la configuration en lecture seule ;
  - les données en lecture / écriture.

# Sécurité de l'image

- Contrôle de l'intégrité du conteneur :
  - stockage ;
  - transport sur le réseau ;
  - vérification avant exécution.

- Distribution des secrets ?
  - Considérer une image de conteneur comme publique ;
  - Ne pas inclure de secrets dans une image de conteneur ;
  - Utilisation de variables d'environnement ;
  - Récupération dynamique des secrets : Vault [88].

# Plan


6 Conteneurs et sécurité

- Sécurité de l’hôte
- Sécurité de l’image
- Sécurité de l’application

# Sécurité des applications : langages

- Une application vulnérable sera toujours vulnérable dans un conteneur ;
- Utiliser des langages de programmations et des Frameworks avec des propriétés intéressantes en terme de sécurité :
  - Rust, Go, Kotlin ;
  - Haskell, OCaml, Java, Elixir, Erlang, Python.

# Sécurité des applications : système de base

- Conteneurs sont souvent créés à partir d'un système de base ;
- La taille n'est pas toujours un facteur de décision (même si l'on cherche souvent à la réduire au maximum) ;
- Choisir un système de base avec des bonnes propriétés de sécurité et correctement supporté :
  - Attention aux images basées sur Alpine :
    - Compatibilité avec musl libc et LibreSSL ;
    - Gestionnaire de paquet spécifique (apk) [89].
  - Les distributions standard (Debian, Ubuntu, Fedora, Red Hat/CentOS) restent une valeur sûre.

# Sécurité des applications : mises à jour

- Conteneur $\Leftrightarrow$ énorme binaire statique ;
- Absolument prévoir un cycle de mise à jour ;
- Intégration Continue (CI) ;
- Outils pour vérifier qu'une image ne contient pas de vulnérabilité déjà connue :
  - Clair [114] ;
  - Open Scap [115].

# Plan


1 Infrastructures

2 Conteneurs

3 systemd

4 Gestionaires de conteneurs

5 SELinux

6 Conteneurs et sécurité

7 Virtualisation et sécurité

- Virtualisation avec KVM/QEMU
- Scénarios d'attaque et vulnérabilités
- Virtulisation et sécurité

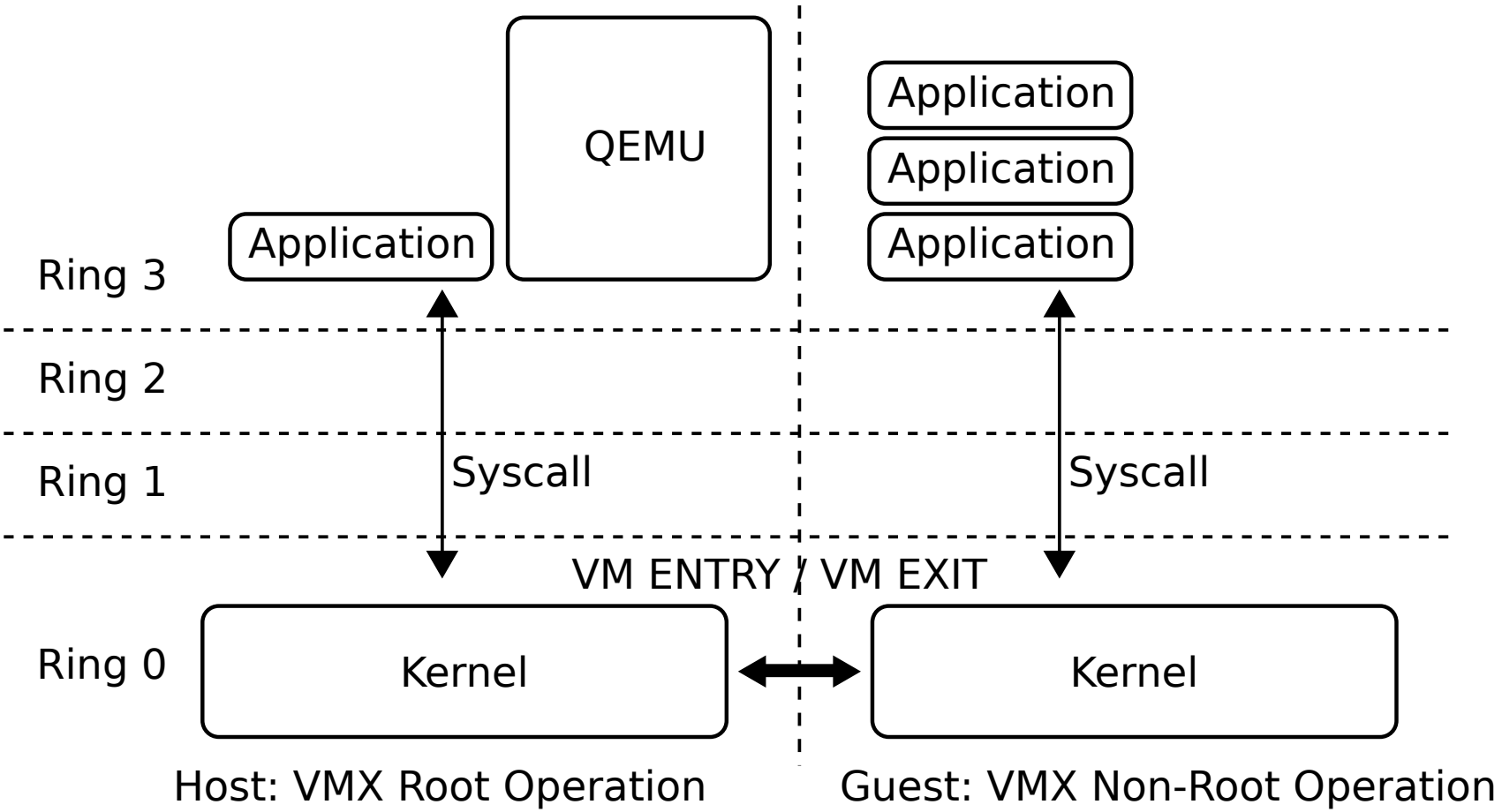
8 Sécurité des infrastructures

# Plan

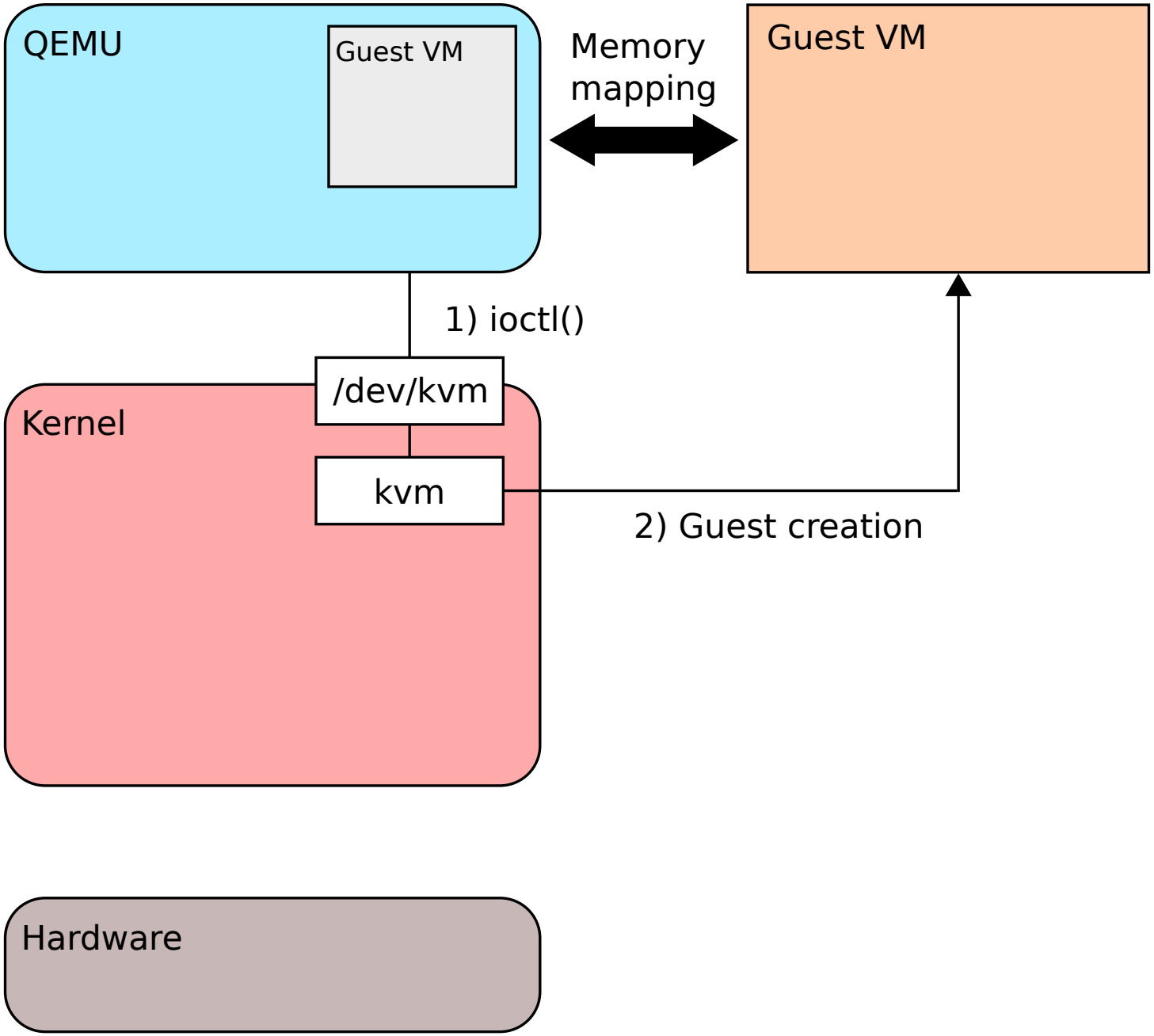
7 Virtualisation et sécurité

- Virtualisation avec KVM/QEMU
- Scénarios d'attaque et vulnérabilités
- Virtulisation et sécurité

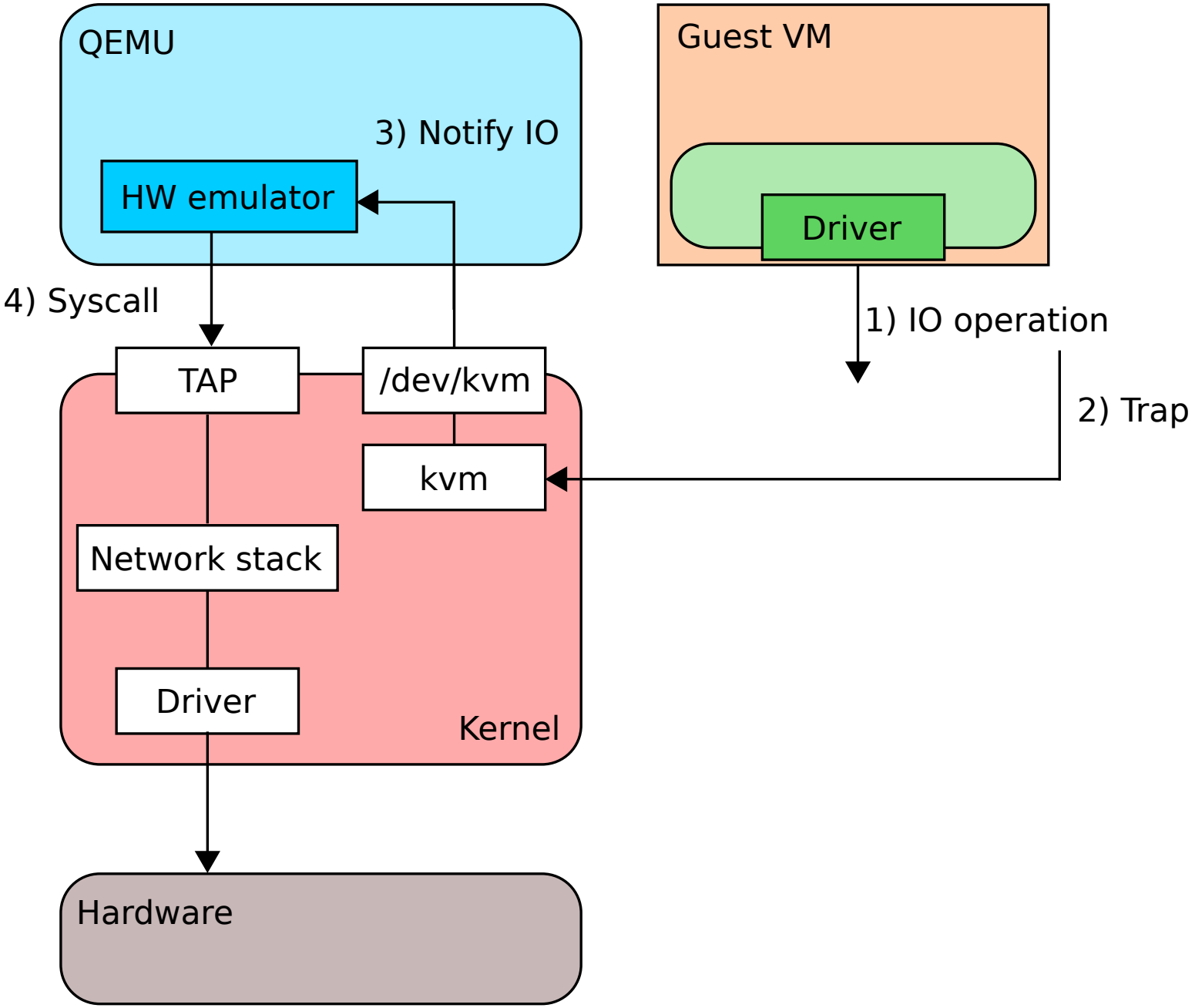
# Principe général : KVM & VT-x (Intel)



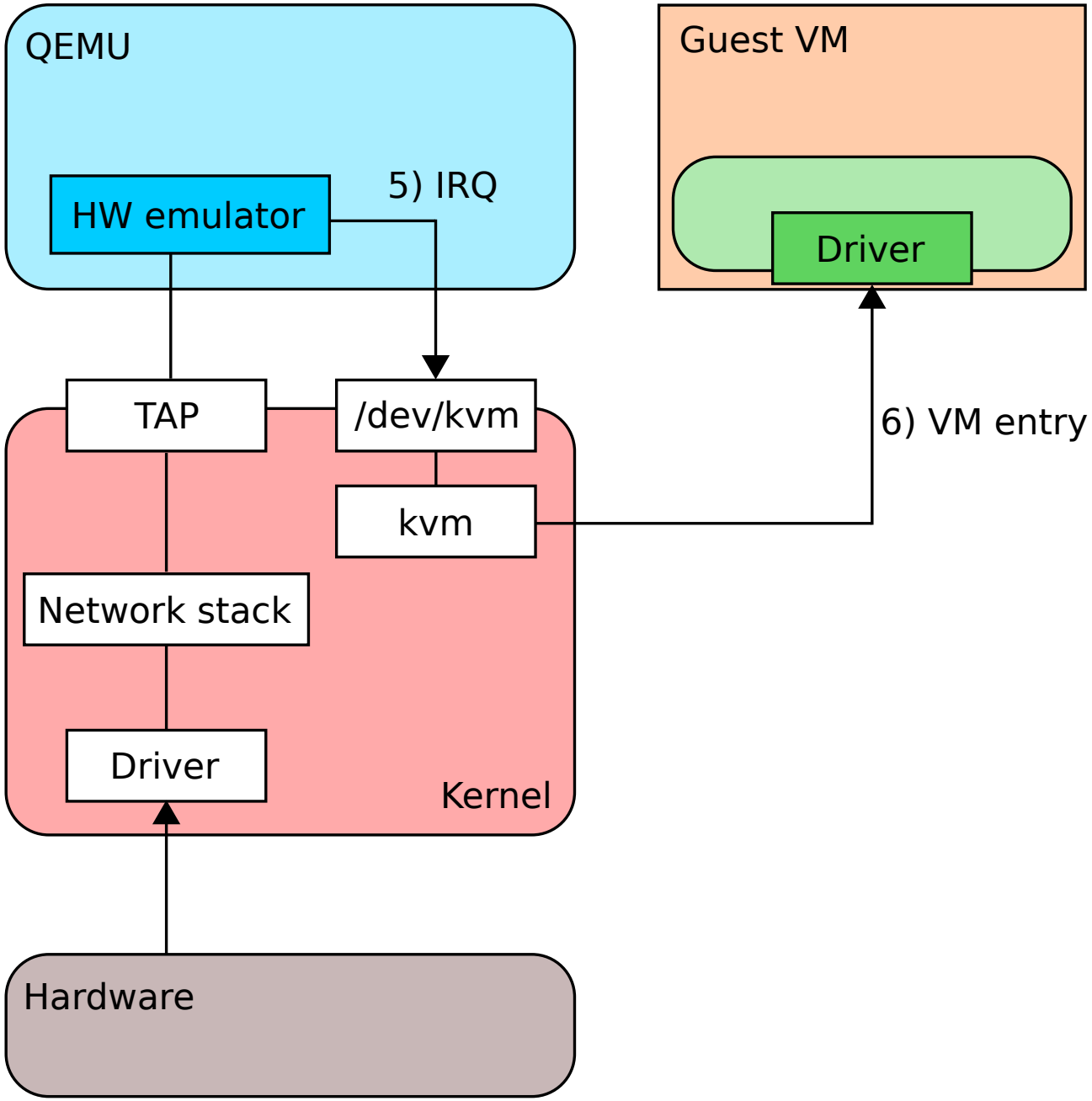
# Création d’une VM avec QEMU



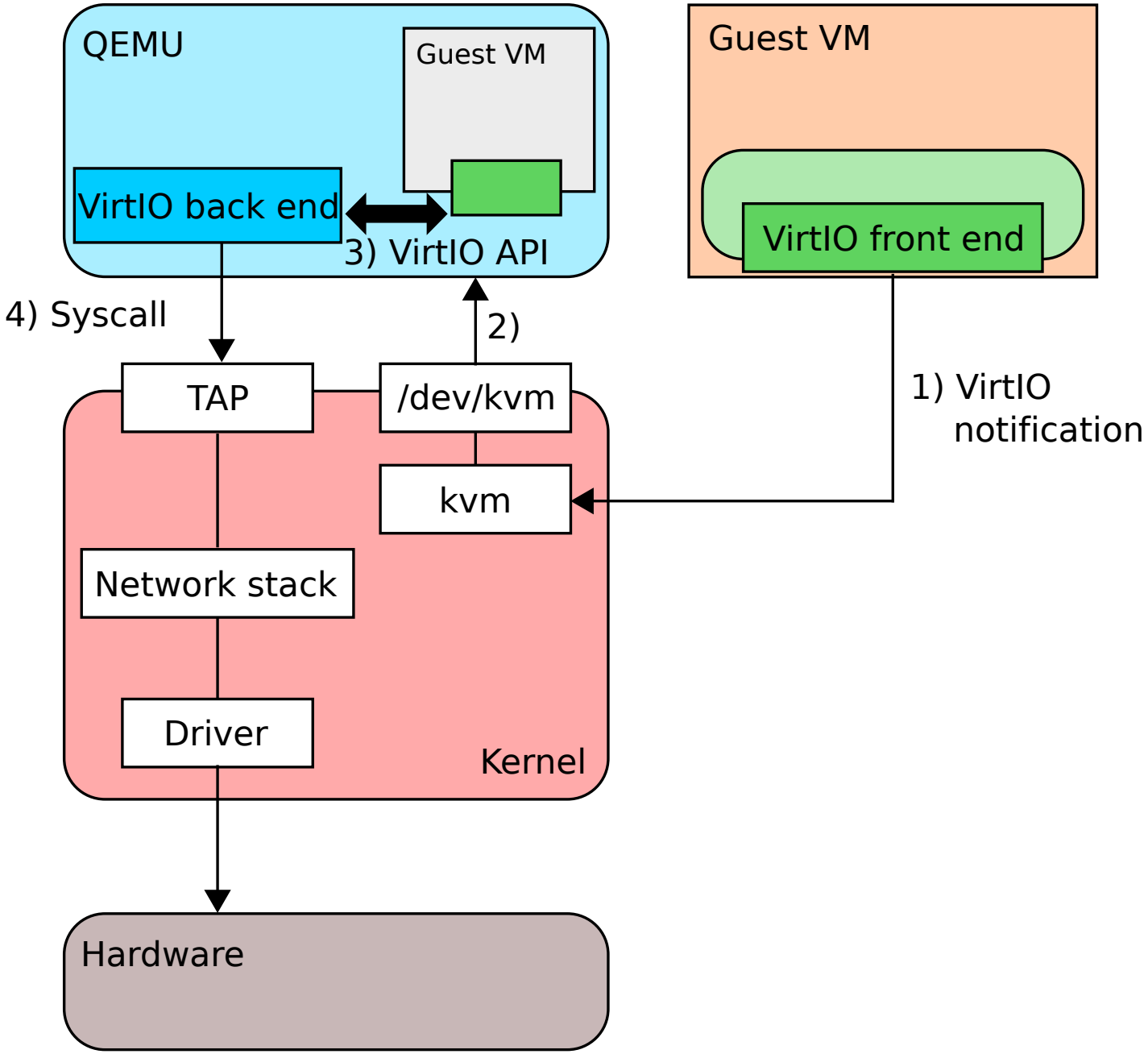
# Matériel émulé (QEMU) I



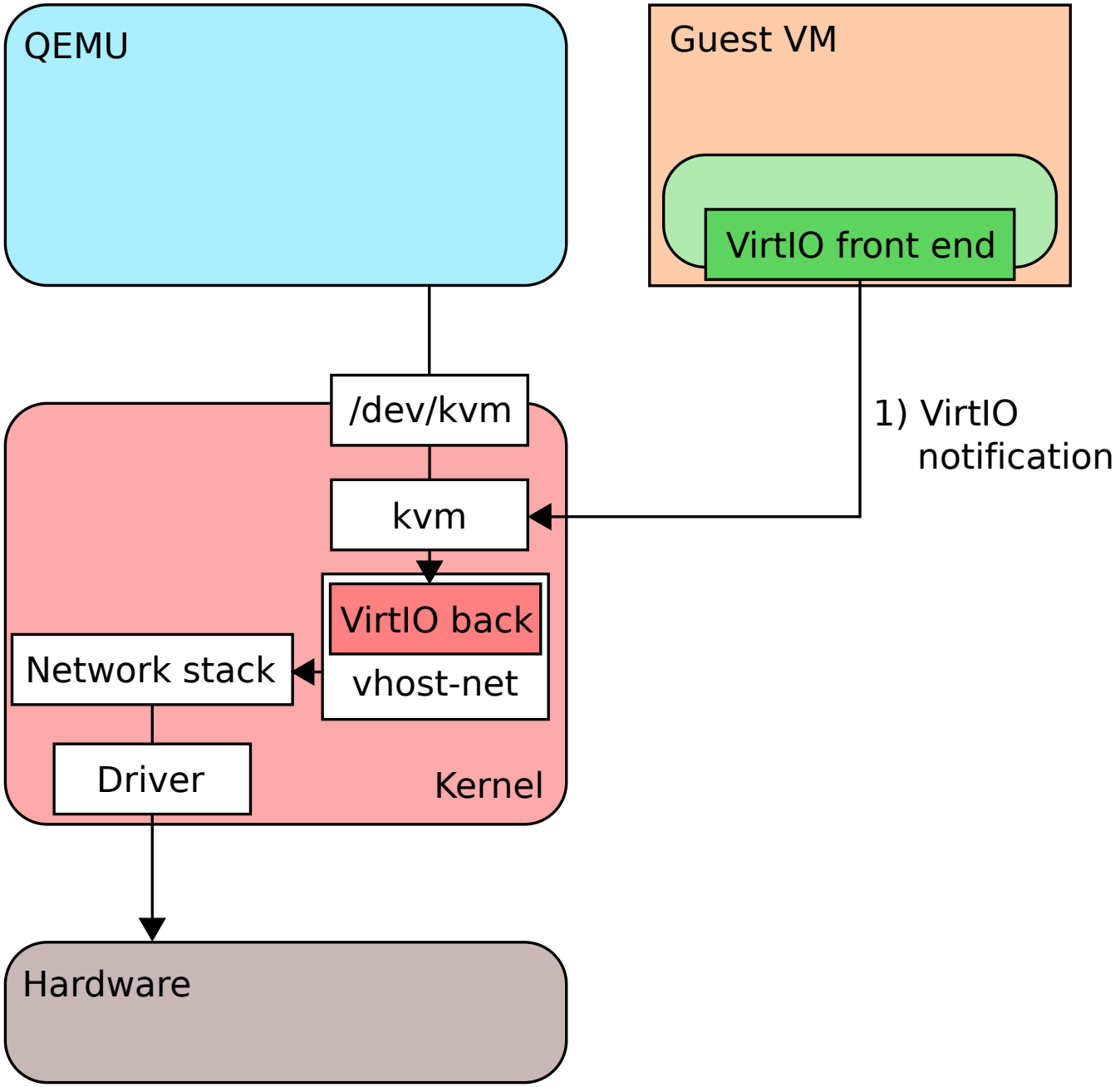
# Matériel émulé (QEMU) II



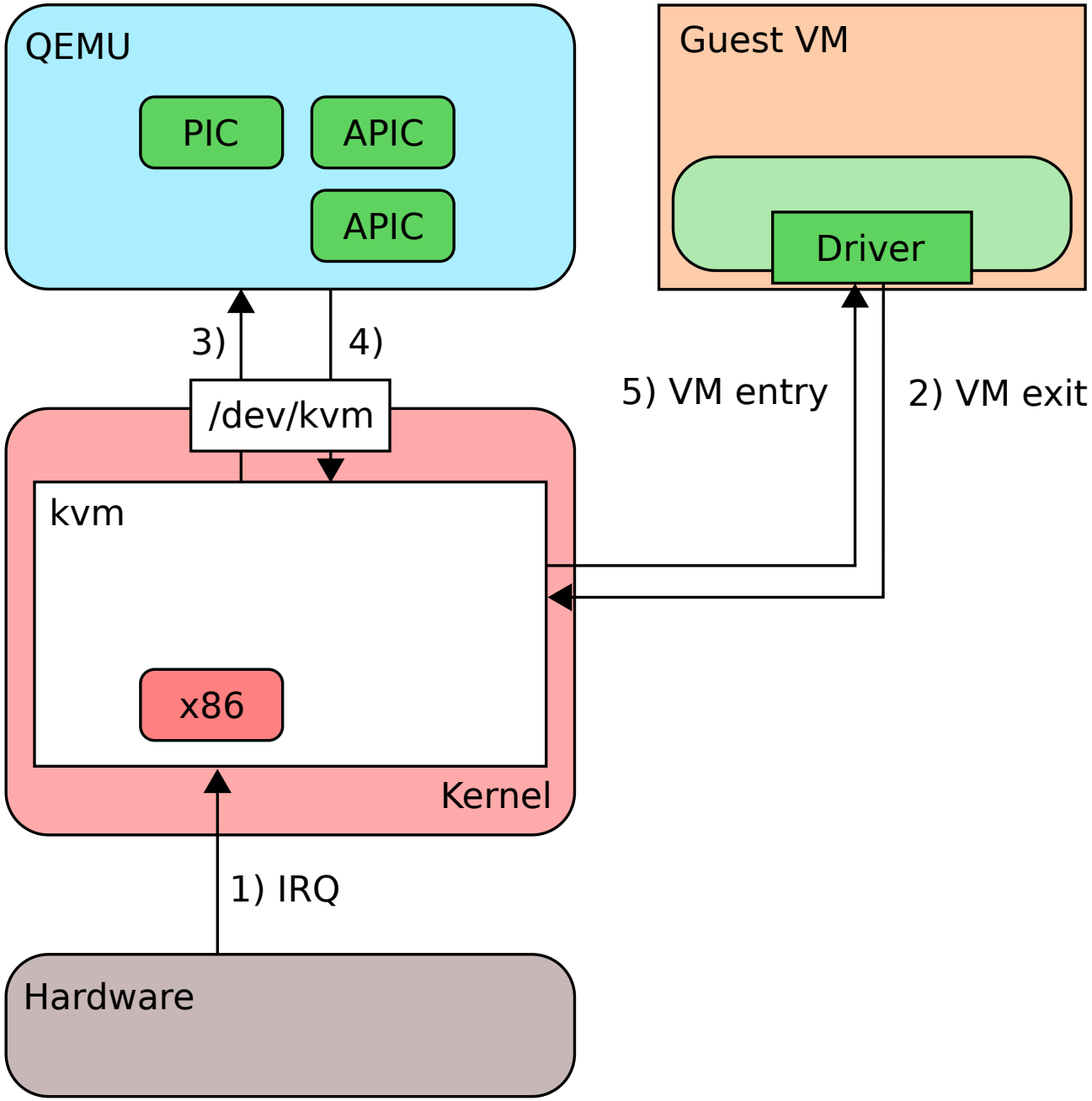
# Matériel virtuel : VirtIO (QEMU)



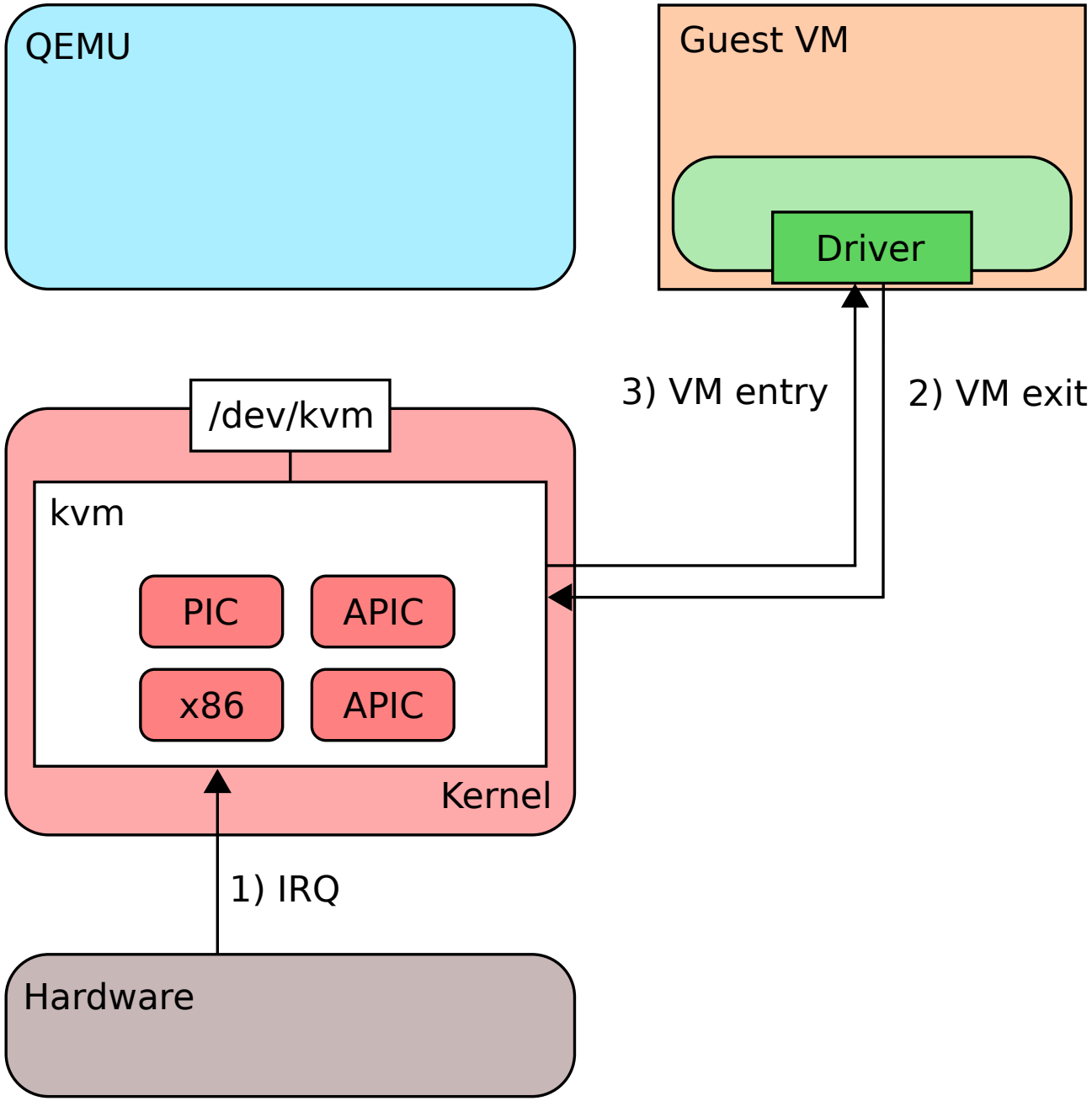
# Matériel virtuel : VirtIO (Kernel)



# Composants émulés (QEMU)



# Composants émulés (Kernel)



# Plan

7 Virtualisation et sécurité

- Virtualisation avec KVM/QEMU
- Scénarios d’attaque et vulnérabilités
- Virtulisation et sécurité

# Virtualisation == sécurité ?

La virtualisation n’est pas un composant de sécurité !

# Virtualisation == sécurité ?

Plus d'isolation, mais :

- Matériel plus compliqué ;
- Plus de versions de systèmes (noyau, espace utilisateur) ;
- Plus de réseaux à gérer ;
- Plus d'interfaces d'administration.

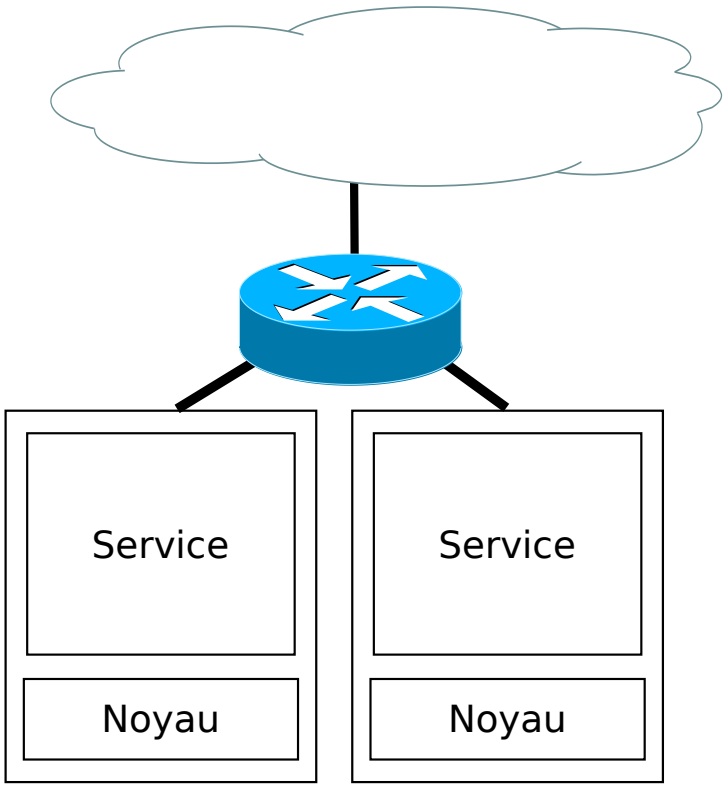
# Virtualisation == sécurité ?

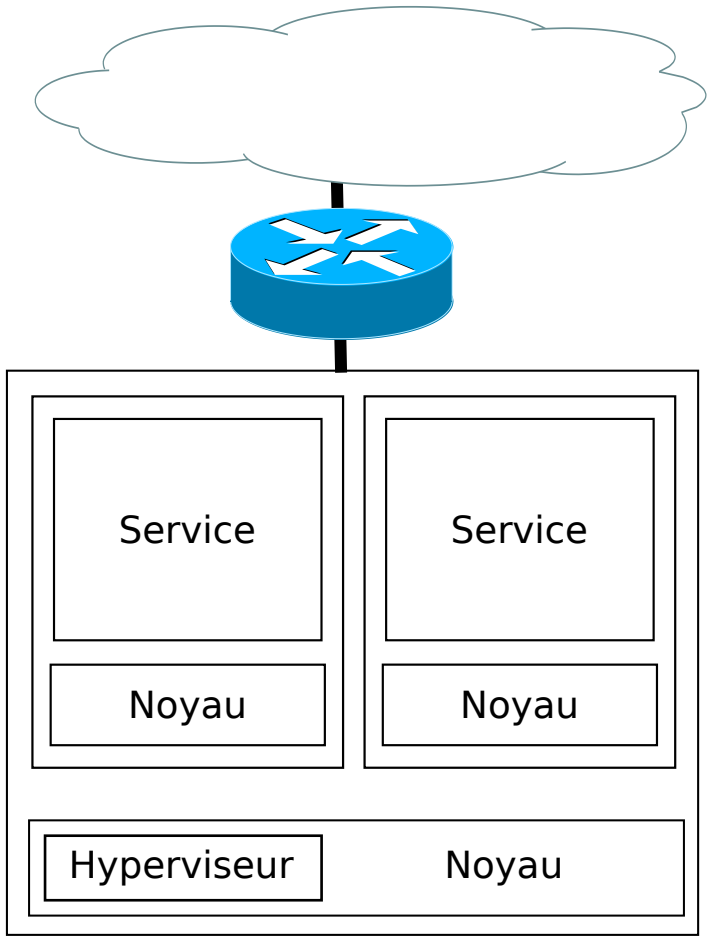
- La virtualisation ne dispense donc pas des autres solutions pour assurer la sécurité ;
- Au contraire, elle en ajoute à appliquer.

# Virtualisation == sécurité ?

Pour s'en convaincre, il faut comparer :

- Services sur des machines physiquement distinctes vs machines virtuelles ;

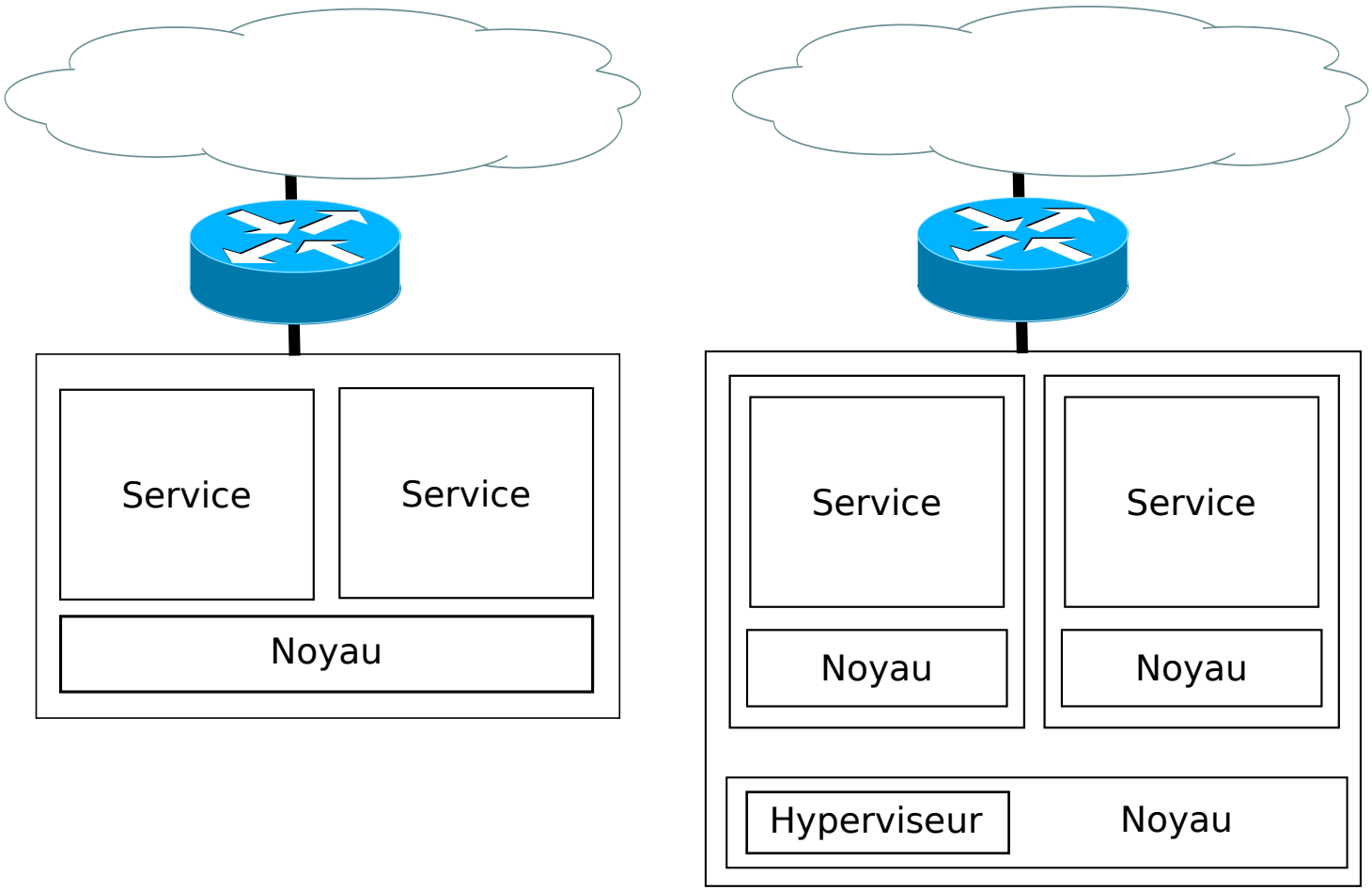


# Virtualisation == sécurité ?

Pour s'en convaincre, il faut comparer :

- Services sur une même machine vs services dans une ou plusieurs machines virtuelles.



# Scénario d’attaque : Machine virtuelle

- La machine virtuelle (le système, les services ou les données) sont visés ;
- Principalement des attaques à distance similaires aux attaques classiques pour les machines physiques ;
- Les machines virtuelles ne sont pas exemptées des contraintes de sécurité appliquées aux machines physique ;

# Scénario d’attaque : Autres VMs

- Corrompre/espionner les autres machines virtuelles sur le même hyperviseur ;

# Scénario d’attaque : Système hôte

- Sortir de la machine virtuelle pour attaquer le système hôte et les autres machines virtuelles ;
- Attaquer ou corrompre l’hyperviseur :
  - faille dans l’hyperviseur ou les drivers virtuels ;
  - faille dans le système hôte.

# Scénario d’attaque : Virtualisation à la volée

- Virtualiser à la volée un système non-virtualisé sans visibilité pour l’utilisateur ;
- Théoriquement possible (Blue Pill [48]) ;
- Invisibilité contestée en pratique : il est très difficile de masquer à un système qu’il est virtualisé.

# Xen

- XSA-109 [55], XSA-123 [56] : élévation de privilège ;
- XSA-148 [57], [58] :
  - Nécessite d'être `root` dans une machine virtuelle PV ;
  - Accès à la totalité de la mémoire du système.
- XSA-155 [59] :
  - Nécessite d'être `root` dans une machine virtuelle PV ou HVM ;
  - Élevation de privilège.

# QEMU

- CVE-2011-1751 (aka Virtunoid) [49] :
  - Bug dans la gestion du temps dans QEMU ;
  - Nécessite d'être `root` la machine virtuelle ;
  - Prise de contrôle de QEMU en Ring 3 mode VMX Root (« sortir » de la machine virtuelle).
- CVE-2015-3456 (aka VENOM) [50] :
  - Bug dans le pilote du contrôler de disquettes émulé par QEMU ;
  - Impacte tous les hyperviseurs ;
  - Activé par défaut dans toutes les machines virtuelles sans possibilité de le désactiver à l'exécution ;
  - Nécessite d'être `root` la machine virtuelle ;
  - Prise de contrôle de QEMU en Ring 3 mode VMX Root (« sortir » de la machine virtuelle).

# KVM

- CVE-2015-0239 : Émulation incomplète de l'instruction SYSENTER [63] ;
- CVE-2014-0049 : Erreur dans l'émulateur x86 [64].

# Hôte : VirtIO

- Faille driver VirtIO - CVE-2011-4127 [65] :
  - Nécessite d'être `root` la machine virtuelle ;
  - Donne l'accès total à un disque physique SCSI de l'hôte car le noyau ne vérifiait pas que les commandes SCSI correspondaient uniquement à des parties accessibles à la VM ;
  - Faille dans le noyau hôte.

# Hôte : système de fichier

- Il est très fortement déconseillé de monter directement le système de fichiers d'une machine virtuelle sur la machine hôte [67] :
  - C'est `mount` qui détermine quel module doit être chargé en fonction du système de fichiers : il peut donc être trompé ;
  - Une faille dans un driver de système de fichier ou dans la couche VFS du noyau Linux peut compromettre toute la machine :
- Il faut donc utiliser d'autres outils qui le font indirectement : `libguestfs`.

# Plan

7 Virtualisation et sécurité

- Virtualisation avec KVM/QEMU
- Scénarios d’attaque et vulnérabilités
- Virtulisation et sécurité

# Risques principaux

- Sécurité de l'hyperviseur :
  - Matériel / Noyau / OS ;
  - Gestionnaire de machines virtuelles (VMM) ;
  - Emulateur / virtualiseur ;
  - Isolation entre les machines virtuelles.
- Sécurité des machines virtuelles ;
  - Noyau / OS ;
  - Applications ;
  - Mise à jour.
- Sécurité des images de machines virtuelles :
  - Distribution et intégrité ;
  - Distribution des secrets.

# Matériel

- Matériel sécurisé ?
- Canaux cachés ;
- Matériel de confiance ;
- Cf. chapitre sur la sécurité des conteneurs.

# Matériel : virtualisation

Surface d’attaque supplémentaire :

- Fonctions de virtualisation dans les processeurs ;
- CPU et MMU : SVM/AMD-V (AMD), VT-x (Intel) ;
- IOMMU : AMD-Vi (AMD), VT-d (Intel).

# IOMMU & PCI passthrouh

- IOMMU intéressante dans tous les cas : contrôle d'accès entre prériphériques PCI-Express ;
- PCI passthrough et SR-IOV à éviter [66] :
  - firmware du matériel accessible ?
  - interfaces cachées ?
  - quelle isolation entre les différents contextes par le matériel ? (exemple : cartes graphiques [44])
  - ...
- Utiliser VFIO.

# Noyau

- Cf. chapitre sur la sécurité des conteneurs ;
- Point commun entre toutes les machines virtuelles ;
- Hyperviseur : module KVM : (~30k SLOC)
  - `/dev/kvm` : interface `iotcl` ;
  - 2 hypercalls : `KVM_HC_VAPIC_POLL_IRQ` ($\Leftrightarrow$ `VM_EXIT`) et `KVM_HC_KICK_CPU`.
- Drivers spécifiques :
  - vhost-net ;
  - vhost-scsi.

# Durcissement KVM

Éviter :

- les drivers dans le noyau : pas de vhost-net/scsi ;
- les émulateurs matériels dans le noyau : pas de PIC, APIC, IOAPIC :
  - À désactiver avec des options pour QEMU [42,43].

# Éviter Xen

- De très nombreuses vulnérabilités ;
- Dernier Cloud provider majeur : Amazon AWS.
- Migration en cours vers KVM (Nitro).

# Noyau Linux : durcissement

- Cf. chapitre sur les conteneurs.

# Système d’exploitation

- Le système d’exploitation hôte de l’hyperviseur est un système classique ;
- Cf. chapitre sur les conteneurs.

# Gestionnaire de machines virtuelles (VMM)

- Le gestionnaire dispose de tous les accès ;
- Contrôle d'accès ;
- Exemple pour libvirt :
  - Accès à la socket libvirt en local ;
  - Accès à distance en TCP.

# Emulateur / virtualiseur

- Émulateur : QEMU (~500k SLOC) ;
- Process en espace utilisateur :
  - Réduction de la surface d'attaque : moins d'émulateurs ;
  - Options de compilation / durcissement ;
  - Utilisation de seccomp-bpf ;
  - Confinement avec SELinux.

# Alternatives à QEMU ?

- lkvm / Native KVM tool [52] :
  - virtualisateur seulement ;
  - supporte seulement les périphériques virtio & guests Linux.
- crosvm (Chrome OS) [51] & Firecracker (AWS) :
  - écrit presque exclusivement en Rust ;
  - virtualisateur x64 seulement ;
  - supporte seulement certains périphériques virtio.
- novm (Google) :
  - écrit en Go ;
  - supporte seulement certains périphériques virtio.
- Google Compute Engine [53] ;
- Amazon Web Services Nitro [54] ;
- NEMU (Intel) : version nettoyée de QEMU.

# Isolation entre les machines virtuelles

- SELinux : confinement des instances de QEMU (sVirt) ;
- Voir chapitre sur la sécurité des conteneurs ;
- Catégories associées aux images disques des machines virtuelles.

# Sécurité des machines virtuelles

- Système d'exploitation complet ;
- Contraintes similaires : mise à jour, durcissement standard, etc.
- Voir chapitre sur la sécurité des conteneurs.

# Sécurité des machines virtuelles : `root`

- Protection du compte `root` et du noyau impérative ;
- La quasi totalité des vulnérabilités dans les hyperviseurs nécessitent d'être `root` (Ring 0) dans la machine virtuelle ;
- Séparation de privilèges et défense en profondeur nécessaire.

# Entropie dans une machine virtuelle

- Beaucoup de composants matériels émulés dans une VM ;
- Comportement très peu aléatoire ;
- Entropie de mauvaise qualité ;
- Solution : VirtIO-RNG.

# Images de machines virtuelles

- Vérifier l'origine et l'intégrité des images de machines virtuelles ;
- Considérer les images de machines virtuelles comme publiques ;
- Cloud-init & Ignition : configuration spécifique à chaque instance ;
- Voir chapitre sur la sécurité des conteneurs pour la gestion des secrets.

# Plan


1 Infrastructures

2 Conteneurs

3 systemd

4 Gestionaires de conteneurs

5 SELinux

6 Conteneurs et sécurité

7 Virtualisation et sécurité

8 Sécurité des infrastructures

- Openstack
- Kubernetes
- Sécurité

# Automatisation / Orchestration

- Gestion automatique des applications ;
- Gestion automatique des ressources ;
- Gestion automatique de la disponibilité.

- Objectif : déléguer les tâches ingrates à des services ;
- Minimiser les temps de panne et augmenter la réactivité.

# Plan

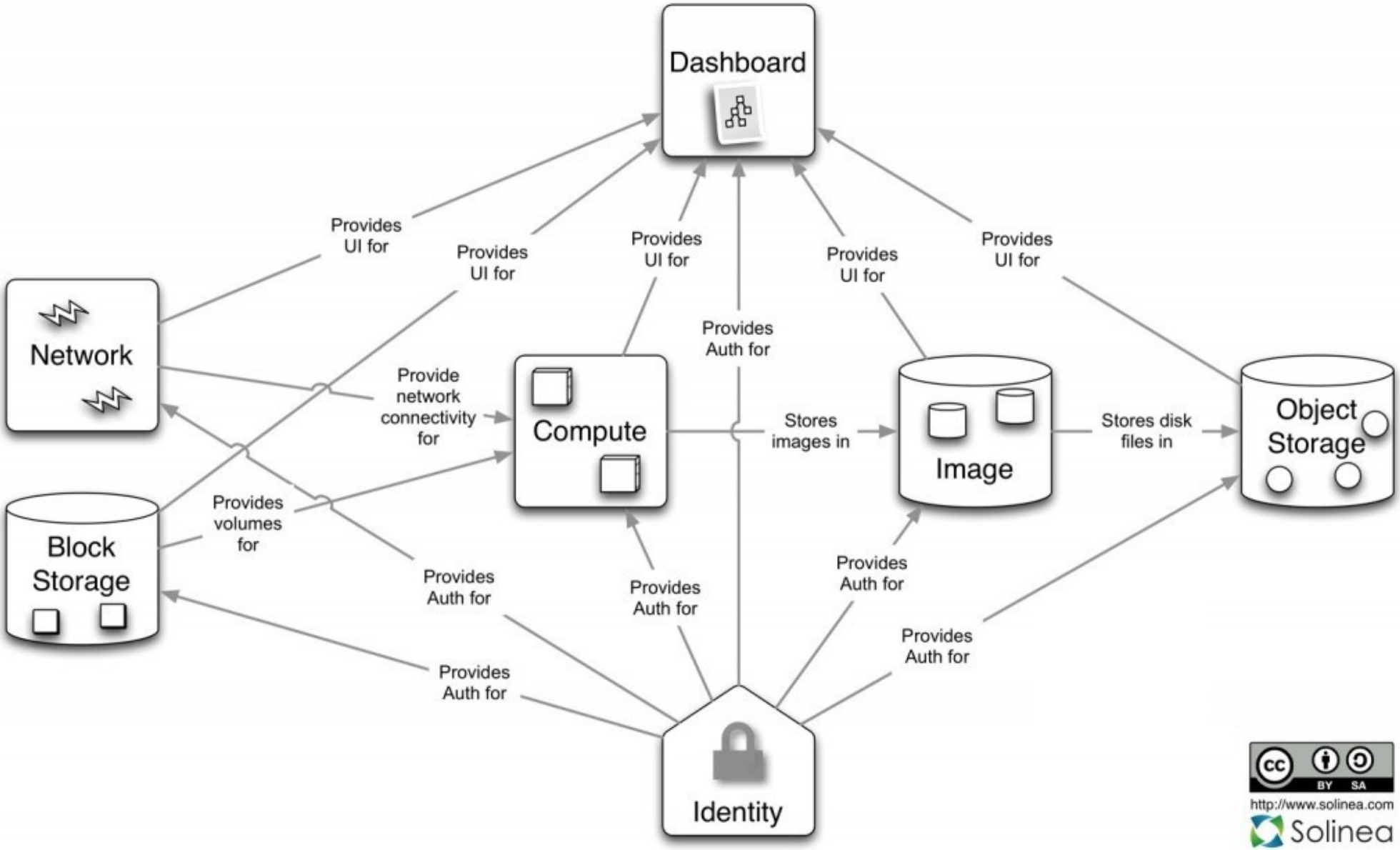
8 Sécurité des infrastructures

- Openstack
- Kubernetes
- Sécurité

# OpenStack

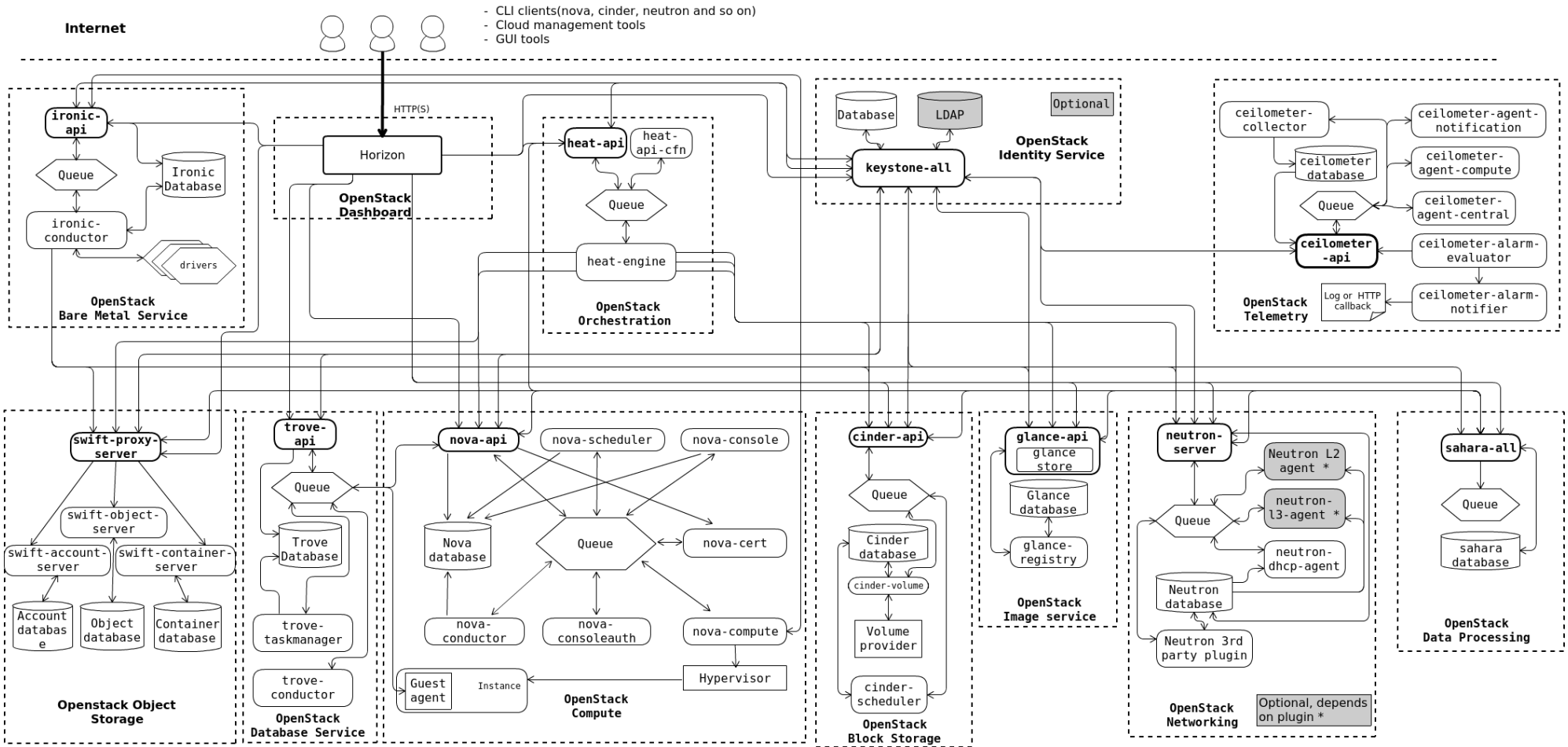
- Projet lancé en 2010 par Rackspace et la NASA ;
- Ensemble de composant pour gérer automatiquement le déploiement de machines virtuelles ou de conteneurs ;
- Nombreux services disponibles ;
- Intégration avec le matériel réseau, stockage, etc.

# OpenStack : services




http://www.solinea.com
Solinea

# OpenStack : services



# Plan


8 Sécurité des infrastructures

- Openstack
- Kubernetes
- Sécurité

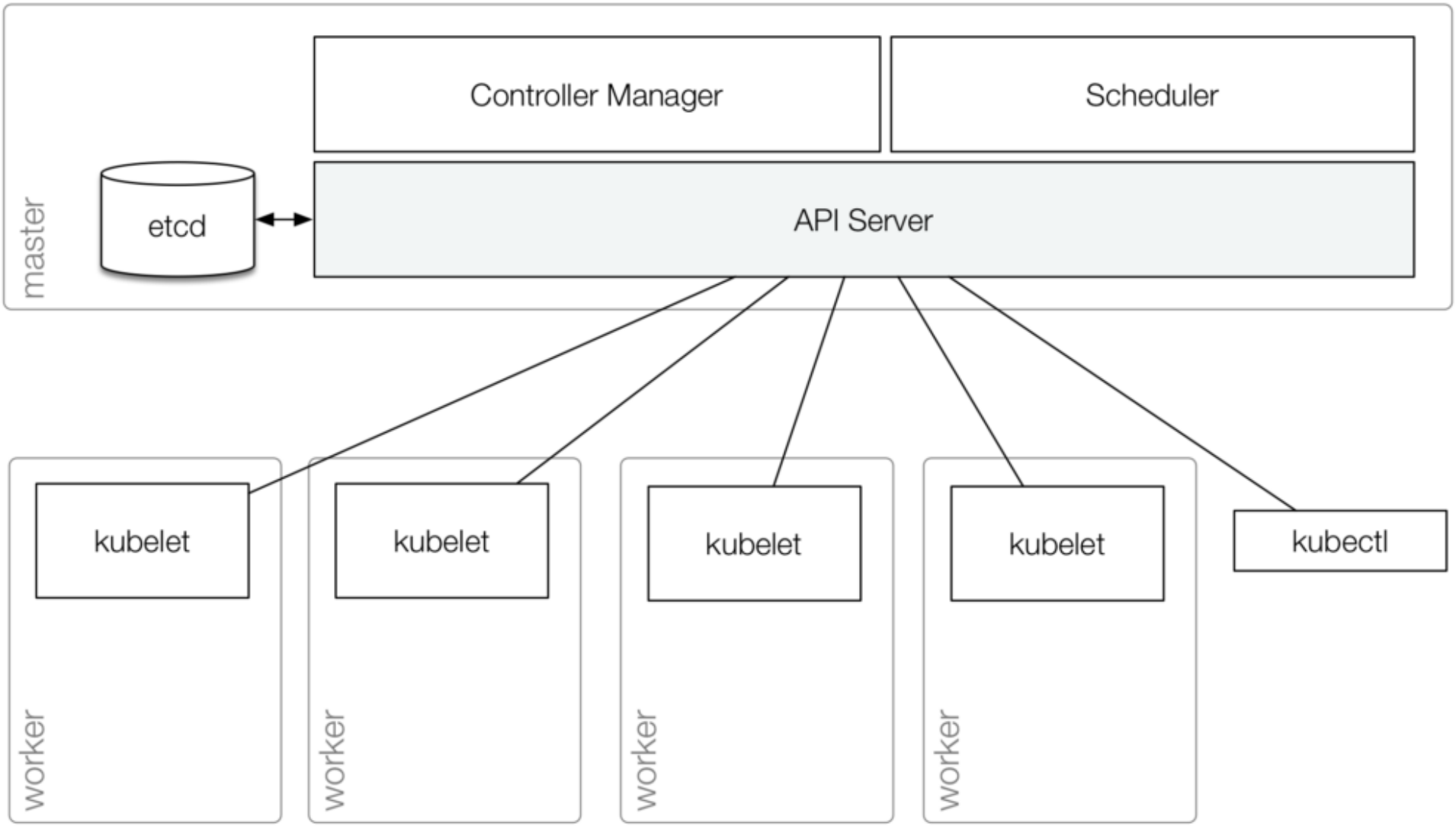
# Kubernetes

- Projet lancé par des ingénieurs chez Google en 2014 [100] ;
- Inspiré de *Borg*, le gestionnaire de conteneur interne Google ;
- Basé sur le concept de Pods (groupe de conteneurs partageant certains namespaces [102], [103]) ;
- Repose sur etcd [104] ;
- Plusieurs distributions (OpenShift (Red Hat [107)], Typhoon, etc.).
- Regroupement de nombreux projets autour de la CNCF [101].

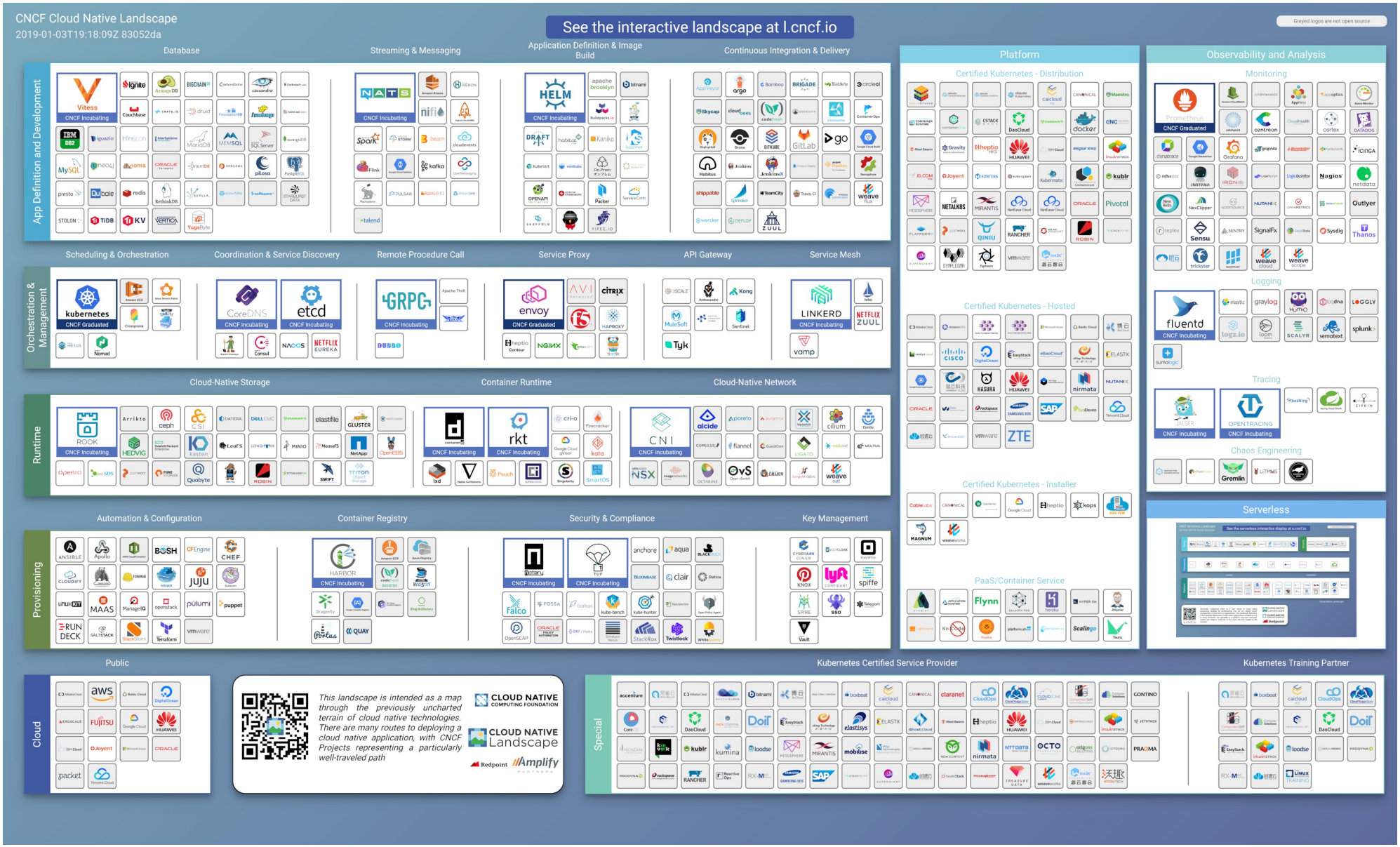
# Kubernetes & Cloud providers

- Versions hébergés / gérées par un Cloud provider de Kubernetes ;
- Google Kubernetes Engine, Amazon EKS, Azure Kubernetes Service, etc.
- Intégré dans les offres de chaque Cloud provider.

# Kubernetes : services



# Cloud Native Computing Landscape



# *Serverless* ou « Fonctions »

## Concept

Exécuter des petites portions de code en réaction à un événement et sans avoir à gérer ou mettre en place une infrastructure complète.

Exemples chez les Cloud providers :

- AWS Lambda, Google Cloud Functions, Azure Functions, Fastly Terrarium, Cloudflare Workers.

Exemples d'implémentations *open source* :

- Knative, Kubeless, Fission, Apache OpenWhisk, IronFunctions, OpenFaaS, Fn Project.

# Serverless

CNCF Serverless Landscape
2019-01-03T19:18:09Z 83052da

See the serverless interactive display at s.cncf.io

Greyed logos are not open source

**Tools**

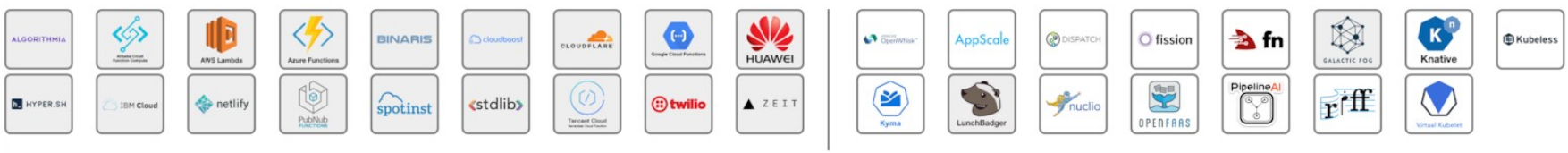

**Security**


**Framework**

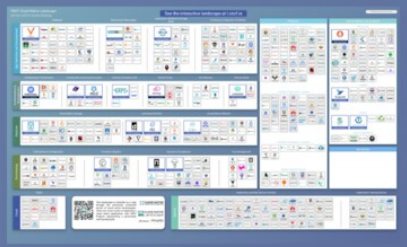

Hosted

Installable

**Platform**




Cloud Native Landscape



s.cncf.io

*Serverless computing refers to a new model of cloud native computing, enabled by architectures that do not require server management to build and run applications. This landscape illustrates a finer-grained deployment model where applications, bundled as one or more functions, are uploaded to a platform and then executed, scaled, and billed in response to the exact demand needed at the moment*

CLOUD NATIVE Landscape

CLOUD NATIVE COMPUTING FOUNDATION

Redpoint

# Plan

8 Sécurité des infrastructures

- Openstack
- Kubernetes
- Sécurité

# Sécurité des infrastructures de virtualisation

- Sécurité des éléments de base de l'infrastructure :
  - Sécurité des hôtes ;
  - Sécurité des machines virtuelles / conteneurs ;
  - Voir les chapitres précédents.
- Sécurité des services de l'infrastructure :
  - Interfaces d'administration ;
  - Interactions entre les services et gestion des réseaux ;
  - Indicateurs, visibilité et inventaire ;
  - Intégration avec les Cloud provider & matériels ;
  - Déploiement, mises à jour.

# Isolation des interfaces d’administration

- Isolation réseau des interfaces d’administrations des hôtes de l’infrastructure.
- Séparer les communications liées à l’administration des autres interactions ;
- Utiliser des protocoles éprouvés (SSH) ;
- Utiliser un réseau physique distinct (ou à minima un réseau logique distinct).

# Isolation des services de l'infrastructure

- La compromission d'un service peut entraîner la compromission de toute l'infrastructure [98] ;
- Séparer les communications entre les services des communications effectuées par les machines virtuelles, conteneurs et applications déployées par les utilisateurs sur l'infrastructure ;
- Utiliser des protocoles éprouvés (TLS 1.2+) ;
- Utiliser un réseau physique distinct (ou à minima un réseau logique distinct).

# Gestion du réseau

- Contrôle des intéractions entre les applications / conteneurs / machines virtuelles à travers le réseau.
- OpenStack Neutron [97].
- Kubernetes :
  - Container Network Interface (CNI) ;
  - flannel (L3, vxlan) [106] ;
  - Cilium (L3-L7, HTTP, BPF) ;
  - Contiv (BGP, vxlan) ;
  - Project Calico (BGP) ;
  - etc. [105].

# Gestion des secrets & identités

- Gestion des secrets : automatiser leur création, stockage, renouvellement et expiration :
  - OpenStack : Anchor, Barbican ;
  - Kubernetes Secrets ou HashiCorp Vault.

- Gestion des identités et des accès :
  - OpenStack : Keystone ;
  - Kubernetes RBAC, Dex.

# Indicateurs, Visibilité et inventaire

- Visibilité sur l'état de l'architecture :
  - performance (Ceilometer, Prometheus, etc.) ;
  - audit (osquery).

# Gestion et administration

- Gestion manuelle impossible ;
- Automatisation obligatoire ;
- Les accès directs en SSH doivent être restreints aux situations critiques et au debug ;

- Déploiement automatisé (Qualification et Prod) ;
- Gestion des services à l'aide d'outils de gestion de configuration (Chef, Puppet, Ansible, etc.)
- Attention : outils non adaptés à la configuration des instances déployées par les utilisateurs sur l'infrastructure.

Kubernetes : Operators

- Logiciel qui se charge de gérer un déploiement dans une infrastructure Kubernetes ;
- Objectif : automatiser l'administration d'un service ;
- Exemples et état des lieux : [113].

# Mises à jour des hôtes

- La mise à jour des hôtes de l'infrastructure doit être non-intéractive, atomique et coordonnées :
  - Systèmes d'exploitation centrés sur l'utilisation de conteneurs ;
  - Pas ou peu de gestion de paquets ;
  - Retour en arrière possible après une mise à jour non fonctionnelle ;
- Exemples :
  - CoreOS [90], Fedora CoreOS & Red Hat Core OS [93] ;
  - openSUSE Kubic ;
  - VMware Photon OS ;
  - Container-Optimized OS (Google Cloud seulement).

# Mise à jour de l'infrastructure ?

- Failles de sécurité inévitables. À prendre en compte ;
- Quel impact sur le fonctionnement du Cloud ?

- OpenStack : concepts d'UnderCloud et d'OverCloud (TripleO : OpenStack On OpenStack) ;
- Kubernetes : Self hosted Kubernetes (« Bootstrap » puis gestion des services de Kubernetes par lui même).

- Infrastructure as Code (Terraform) : Décrire l'infrastructure sous forme de code pour pouvoir la déployer / redéployer automatiquement.

# Cloud providers

- Pas d'accès direct à l'infrastructure ;
- Cloud provider responsable de la sécurité de l'infrastructure mise à disposition des utilisateurs ;
- Utilisateurs responsable de la bonne configuration des mécanismes de contrôle d'accès ;
- Risque principal financier !
- Protection des comptes administrateurs à l'aide de l'authentification à deux facteurs / deux étapes, avec par ordre de préférence :
  - Token hardware U2F (FIDO 1 & 2 : Yubikey, NitroKey, etc.) ;
  - TOTP hardware ou software.
- Banir l'usage du SMS.

# Sécurité OpenStack

Ressources :

- OpenStack Security Guide [95] ;
- OpenStack Administrator Guide [96] ;
- Security and Hardening Guide (Red Hat) [99].

# Sécurité Kubernetes

- Alternative à Docker :
  - CRI-O (stabilité & sécurité).
- TLS automatique pour les communications entre conteneurs :
  - Istio, Consul, etc.
- Ressources :
  - Pod Security Policy [111] :
    - Obliger tous les conteneurs à tourner en !root ;
    - Obliger tous les conteneurs à tourner avec le / en RO.
  - Securing a Cluster [108] ;
  - Kubernetes The Hard Way [109] ;
  - Container Security Guide (Red Hat) [110] ;
  - OpenShift Container Security Guide (Red Hat) [112].

# Références I


1. xfs_quota(8) : http://man7.org/linux/man-pages/man8/xfs_quota.8.html
2. Control Groups Resource Management : http://libvirt.org/cgroups.html
3. Control Group v2 : https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/Documentation/cgroup-v2.txt
4. CVE request Linux kernel : ns : user namespaces panic : http://www.openwall.com/lists/oss-security/2015/05/29/5
5. Linux kernel source: commit 51dfcb076d1e1ce7006aa272cb7c4514740c7e47
6. Linux kernel source: commit 87c31b39abcb6fb6bd7d111200c9627a594bf6a9

# Références II


7. Linux namespaces : It is possible to escape from bind mounts : http://www.openwall.com/lists/oss-security/2015/04/03/7
8. User namespaces + overlayfs = root privileges : https://lwn.net/Articles/671641/
9. Anatomy of a user namespaces vulnerability : https://lwn.net/Articles/543273/
10. CVE-2015-1328 : incorrect permission checks in overlayfs, ubuntu local root : http://seclists.org/oss-sec/2015/q2/717
11. /proc & /sys : Enforcing mount options for sysfs and proc : https://lwn.net/Articles/647757/

# Références III


12. Petit état de l'art des systèmes d'initialisation : http://linuxfr.org/news/petit-etat-de-l-art-des-systemes-d-initialisation-1
13. The Biggest Myths about systemd : http://0pointer.de/blog/projects/the-biggest-myths.html
14. Red Hat Summit 2013 - Getting Ready for systemd : https://access.redhat.com/site/videos/403833
15. systemd Freedesktop.org Wiki : http://www.freedesktop.org/wiki/Software/systemd/
16. Journal File Format : http://www.freedesktop.org/wiki/Software/systemd/journal-files/
17. Durcissement système à l'aide de systemd, Timothée Ravier, SSTIC 2017 : https://www.sstic.org/2017/presentation/durcissement_systeme_avec_systemd/

# Références IV


18. Linux Kernel Exploitation - Earning Its Pwnie a Vuln at a Time : https://jon.oberheide.org/files/source10-linuxkernel-jonoberheide.pdf
19. Exploiting the Linux kernel : Measures and countermeasures : https://jon.oberheide.org/files/syscan12-exploitinglinux.pdf
20. Dirty CoW : https://dirtycow.ninja/
21. rkt : https://coreos.com/blog/rocket/
22. Container mechanics in rkt and Linux : http://events.linuxfoundation.org/sites/events/files/slides/Container%20mechanics%20in%20rkt%20and%20Linux.pdf
23. rkt architecture : https://coreos.com/rkt/docs/latest/devel/architecture.html

# Références V


24. App Container - Rocket : https://www.socallinuxexpo.org/sites/default/files/presentations/appc%20%2B%20rocket%20(SCALE%2013x).pdf
25. Using Virtual Machines to Improve Container Security with rkt v0.8.0 : https://coreos.com/blog/rkt-0.8-with-new-vm-support/
26. App Container basics : https://coreos.com/rkt/docs/latest/app-container.html
27. Using rkt with systemd : https://coreos.com/rkt/docs/latest/using-rkt-with-systemd.html
28. Image Fetching Behavior : https://coreos.com/rkt/docs/latest/image-fetching-behavior.html
29. rkt vs other projects : https://coreos.com/rkt/docs/latest/rkt-vs-other-projects.html

# Références VI


30. LXC CVE-2013-6441 : http://www.cvedetails.com/cve/CVE-2013-6441/
31. Security issue in LXC (CVE-2015-1335) : http://www.openwall.com/lists/oss-security/2015/09/29/4
32. Before you initiate a “docker pull” : https://securityblog.redhat.com/2014/12/18/before-you-initiate-a-docker-pull/
33. Docker 1.8 : https://blog.docker.com/2015/08/content-trust-docker-1-8/
34. The Update Framework (TUF) : https://theupdateframework.github.io/
35. Docker Registry : https://github.com/docker/distribution

# Références VII


36. Over 30% of Official Images in Docker Hub Contain High Priority Security Vulnerabilities : http://www.banyanops.com/blog/analyzing-docker-hub/
37. Docker 1.3.2 - Security Advisory [24 Nov 2014] : http://www.openwall.com/lists/oss-security/2014/11/24/5
38. Docker 1.3.3 - Security Advisory [11 Dec 2014] : https://groups.google.com/forum/#!msg/docker-user/nFAz-B-n4Bw/0wr3wvLsnUwJ
39. Docker 1.6.1 - Security Advisory [150507] : http://seclists.org/fulldisclosure/2015/May/28
40. False Boundaries and Arbitrary Code Execution : https://forums.grsecurity.net/viewtopic.php?f=7&t=2522
41. CAP_SYS_ADMIN : the new root : https://lwn.net/Articles/486306/

# Références VIII


42. KVM Security Improvements, Andrew Honig, KVM Forum 2014 : http://www.linux-kvm.org/images/f/f6/01x02-KVMHardening.pdf
43. Performant Security Hardening, Steve Rutherford, KVM Forum 2016 : https://www.linux-kvm.org/images/3/3d/01x02-Steve_Rutherford-Performant_Security_Hardening_of_KVM.pdf
44. XDC2012 : Graphics stack security : https://lwn.net/Articles/517375/
45. Reverse Engineering Intel DRAM Addressing and Exploitation : http://arxiv.org/abs/1511.08756
46. Cross Processor Cache Attacks : http://eprint.iacr.org/2015/1155

# Références IX


47. RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis : http://arstechnica.com/security/2013/12/new-attack-steals-e-mail-decryption-keys-by-capturing-compu
48. Introducing Blue Pill : http://theinvisiblethings.blogspot.com/2006/06/introducing-blue-pill.html
49. Virtunoid : A KVM Guest $\rightarrow$ Host privilege escalation exploit : http://media.blackhat.com/bh-us-11/Elhage/BH_US_11_Elhage_Virtunoid_WP.pdf
50. VENOM, don't get bitten : https://securityblog.redhat.com/2015/05/13/venom-dont-get-bitten/
51. crosvm - The Chrome OS Virtual Machine Monitor : https://chromium.googlesource.com/chromiumos/platform/crosvm/

# Références X


52. Stand-alone KVM tool : https://git.kernel.org/pub/scm/linux/kernel/git/will/kvmtool.git
53. 7 ways we harden our KVM hypervisor at Google Cloud : security in plaintext : https://cloudplatform.googleblog.com/2017/01/7-ways-we-harden-our-KVM-hypervisor-at-Google-Cloud-securi html
54. AWS EC2 Virtualization 2017 : Introducing Nitro :http://www.brendangregg.com/blog/2017-11-29/aws-ec2-virtualization-2017.html
55. Xen Security Advisory CVE-2014-8594 / XSA-109 : http://xenbits.xen.org/xsa/advisory-109.html
56. Xen Security Advisory CVE-2015-2151 / XSA-123 : http://xenbits.xen.org/xsa/advisory-123.html

# Références XI


57. Xen Security Advisory CVE-2015-7835 / XSA-148 : http://xenbits.xen.org/xsa/advisory-148.html

58. Qubes Security Bulletin #22 : https://github.com/QubesOS/qubes-secpack/blob/master/QSBs/qsb-022-2015.txt

59. Xen Security Advisory CVE-2015-8550 / XSA-155 : http://xenbits.xen.org/xsa/advisory-155.html

60. Meltdown and Spectre : https://meltdownattack.com/

61. The Intel SYSRET privilege escalation : http://blog.xen.org/index.php/2012/06/13/the-intel-sysret-privilege-escalation/

62. Attacking SMM Memory via Intel CPU Cache Poisoning : http://theinvisiblethings.blogspot.fr/2009/03/attacking-smm-memory-via-intel-cpu.html

# Références XII


63. KVM SYSENTER emulation vulnerability - CVE-2015-0239 : http://www.openwall.com/lists/oss-security/2015/01/27/6

64. Linux kernel source: commit a08d3b3b99efd509133946056531cdf8f3a0c09b

65. CVE-2011-4127 : privilege escalation from qemu / KVM guests : http://rwmj.wordpress.com/2011/12/22/cve-2011-4127-privilege-escalation-from-qemu-kvm-guests/

66. Network Function Virtualization, Packet Processing Performance of Virtualized Platforms with Linux and Intel Architecture : https://networkbuilders.intel.com/docs/network_builders_RA_NFV.pdf

# Références XIII


67 A reminder why you should never mount guest disk images on the host OS : https://www.berrange.com/posts/2013/02/20/a-reminder-why-you-should-never-mount-guest-disk-images-o

68 FUSE : Filesystem in Userspace : http://en.wikipedia.org/wiki/Filesystem_in_Userspace

69 Problématiques de sécurité associées à la virtualisation des systèmes d'information : http://www.ssi.gouv.fr/guide/problematiques-de-securite-associees-a-la-virtualisation-des-s

70 Recommandations de sécurité relatives à un système GNU/Linux : http://www.ssi.gouv.fr/guide/recommandations-de-securite-relatives-a-un-systeme-gnulinux

# Références XIV


71 Recommandations de configuration matérielle de postes clients et serveurs x86 : http://www.ssi.gouv.fr/administration/guide/recommandations-de-configuration-materielle-de-postes-client

72 Recommandations relatives à l'administration sécurisée des systèmes d'information : http://www.ssi.gouv.fr/administration/guide/securiser-ladministration-des-systemes-dinformation/

73 Recommandations de sécurité pour la mise en oeuvre d'un système de journalisation : http://www.ssi.gouv.fr/administration/guide/recommandations-de-securite-pour-la-mise-en-oeuvre-dun-sys

74 NIST Application Container Security Guide : https://www.nist.gov/publications/application-container-security-guide

# Références XV


75 Security Assurance Requirements for Linux Application Container Deployments : https://csrc.nist.gov/publications/detail/nistir/8176/final

76 NIST Guidance on Application Container Security : https://csrc.nist.gov/publications/detail/itl-bulletin/2017/10/application-container-security/final

77 Container Security with SELinux and CoreOS : https://coreos.com/blog/container-security-selinux-coreos.html

78 Announcing rkt v0.7.0, featuring a new build system, SELinux and more : https://coreos.com/blog/rkt-0.7.0-with-selinux-and-new-build-system.html

# Références XVI


79 Docker SELinux Security Policy : https://access.redhat.com/documentation/en/red-hat-enterprise-linux-atomic-host/7/container-security-guide/chapter-6-docker-selinux-security-policy

80 State of SELinux, Paul Moore, LSS 2017 : http://www.paul-moore.com/docs/lss-state_of_selinux-pmoore-092017-r3.pdf

81 Honey, I Shrunk the Attack Surface, Adventures in Android Security Hardening, Nick Kralevich, Black Hat 2017 : https://www.blackhat.com/docs/us-17/thursday/us-17-Kralevich-Honey-I-Shrunk-The-Attack-Surface-Adventure
pdf

# Références XVII


82 What roles do DAC (file permissions), ACL and MAC (SELinux) play in Linux file security ? : https://unix.stackexchange.com/questions/16828/what-roles-do-dac-file-permissions-acl-and-mac-selinux-play-in

83 Where does CIL play in the SELinux system ? : http://blog.siphos.be/2015/06/where-does-cil-play-in-the-selinux-system/

84 Docker Security : Using Containers Safely in Production : https://www.openshift.com/promotions/docker-security.html

85 Docker security : https://docs.docker.com/engine/articles/security/

# Références XVIII


86 Why we don’t let non-root users run Docker in CentOS, Fedora, or RHEL : http://www.projectatomic.io/blog/2015/08/why-we-dont-let-non-root-users-run-docker-in-centos-fedora-o

87 Containers & Docker : How Secure Are They ? : http://blog.docker.io/2013/08/containers-docker-how-secure-are-they/

88 HashiCorp Vault : https://www.vaultproject.io

89 Remote Code Execution in Alpine Linux : https://justi.cz/security/2018/09/13/alpine-apk-rce.html

90 CoreOS : https://coreos.com/

91 What Trusted Computing Means to Users of CoreOS and Beyond : https://coreos.com/blog/coreos-trusted-computing.html

# Références XIX


92. Making Sense of Container Standards and Foundations : OCI, CNCF, appc and rkt : https://coreos.com/blog/making-sense-of-standards.html
93. Project Atomic : http://www.projectatomic.io/
94. Docker Compose : https://docs.docker.com/compose/
95. OpenStack Security Guide : https://docs.openstack.org/security-guide/
96. OpenStack Administrator Guides : https://docs.openstack.org/admin/
97. There's Real Magic behind OpenStack Neutron : https://pinrojas.com/2014/07/29/theres-real-magic-behind-openstack-neutron/

# Références XX


98 OpenStack logical architecture : https://docs.openstack.org/install-guide/get-started-logical-architecture.html

99 Security and Hardening Guide, Red Hat : https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/12/html/security_and_hardening_guide/

100 Kubernetes : http://kubernetes.io/

101 Cloud Native Computing Foundation : https://www.cncf.io/

102 Kubernetes user guide - Pods : http://kubernetes.io/v1.1/docs/user-guide/pods.html

103 App Container Spec - Pods : https://github.com/appc/spec/blob/master/spec/pods.md#app-container-pods-pods

104 etcd : https://coreos.com/etcd/

# Références XXI


[05] Kubernetes Cluster Networking : https://kubernetes.io/docs/concepts/cluster-administration/networking/

[06] flannel : https://coreos.com/flannel/docs/latest/

[07] OpenShift Origin : https://www.openshift.org/

[08] Kubernetes - Securing a Cluster : https://kubernetes.io/docs/tasks/administer-cluster/securing-a-cluster/

[09] Kubernetes The Hard Way, Kelsey Hightower : https://github.com/kelseyhightower/kubernetes-the-hard-way

[10] Container Security Guide, Red Hat : https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html-single/container_security_guide/index

# Références XXII


11. Pod Security Policy : https://kubernetes.io/docs/concepts/policy/pod-security-policy/
12. OpenShift Container Security Guide : https://docs.openshift.com/container-platform/3.7/security/index.html
13. Kubernetes Operators : https://kubedex.com/operators/
14. CoreOS Introduces Clair : Open Source Vulnerability Analysis for your Containers : https://coreos.com/blog/vulnerability-analysis-for-containers/
15. Security compliance of RHEL7 Docker containers : https://www.open-scap.org/resources/documentation/security-compliance-of-rhel7-docker-containers/