le cnam

GLG105: Génie logiciel

Pile de virtualisation du cloud

November 10, 2025

Jacopo Bufalino (CNAM)

Survol

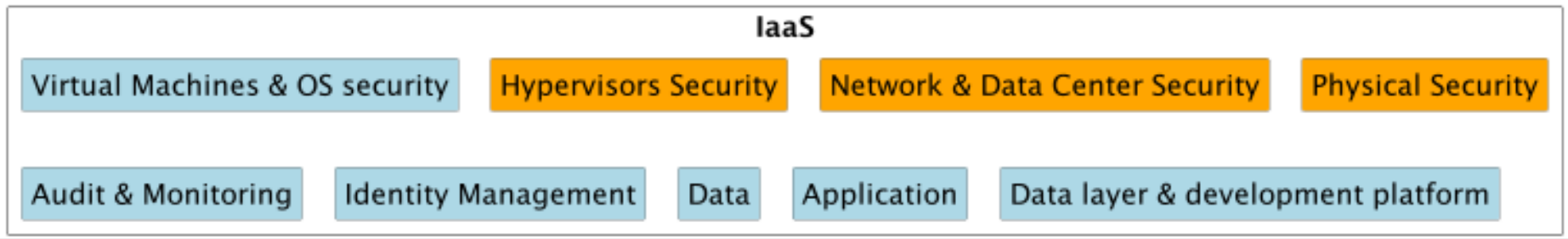
Survol

Ce cours aborde :

- Linux comme système d'exploitation principal pour les environnements cloud
- Les machines virtuelles (VMs)
- Les conteneurs et' leur communication

Les conteneurs seront le sujet du premier TP

Rappel : modèle de responsabilité partagée

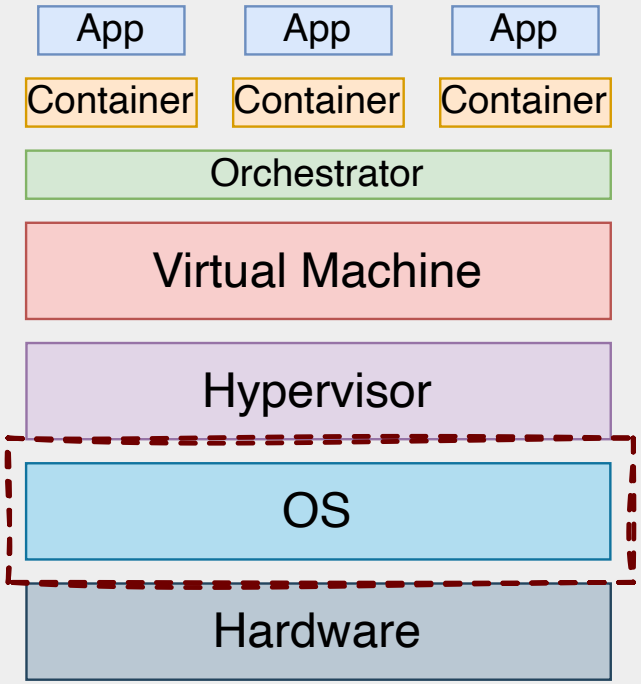


Les systèmes d'exploitation, machines virtuelles et plateformes de développement (conteneurs) sont la responsabilité de l'utilisateur.

Linux

Linux – introduction

- Linux est un système d'exploitation open source, publié pour la première fois en 1991 par Linus Torvalds.
- C'est le système le plus populaire pour l'informatique en nuage et la conteneurisation, pour deux raisons principales :
  - Personnalisation : installation et configuration faciles (logiciels libres et open source – FOSS)
  - Sécurité / isolation : pierre angulaire des environnements multi-locataires comme le cloud



Architecture Linux

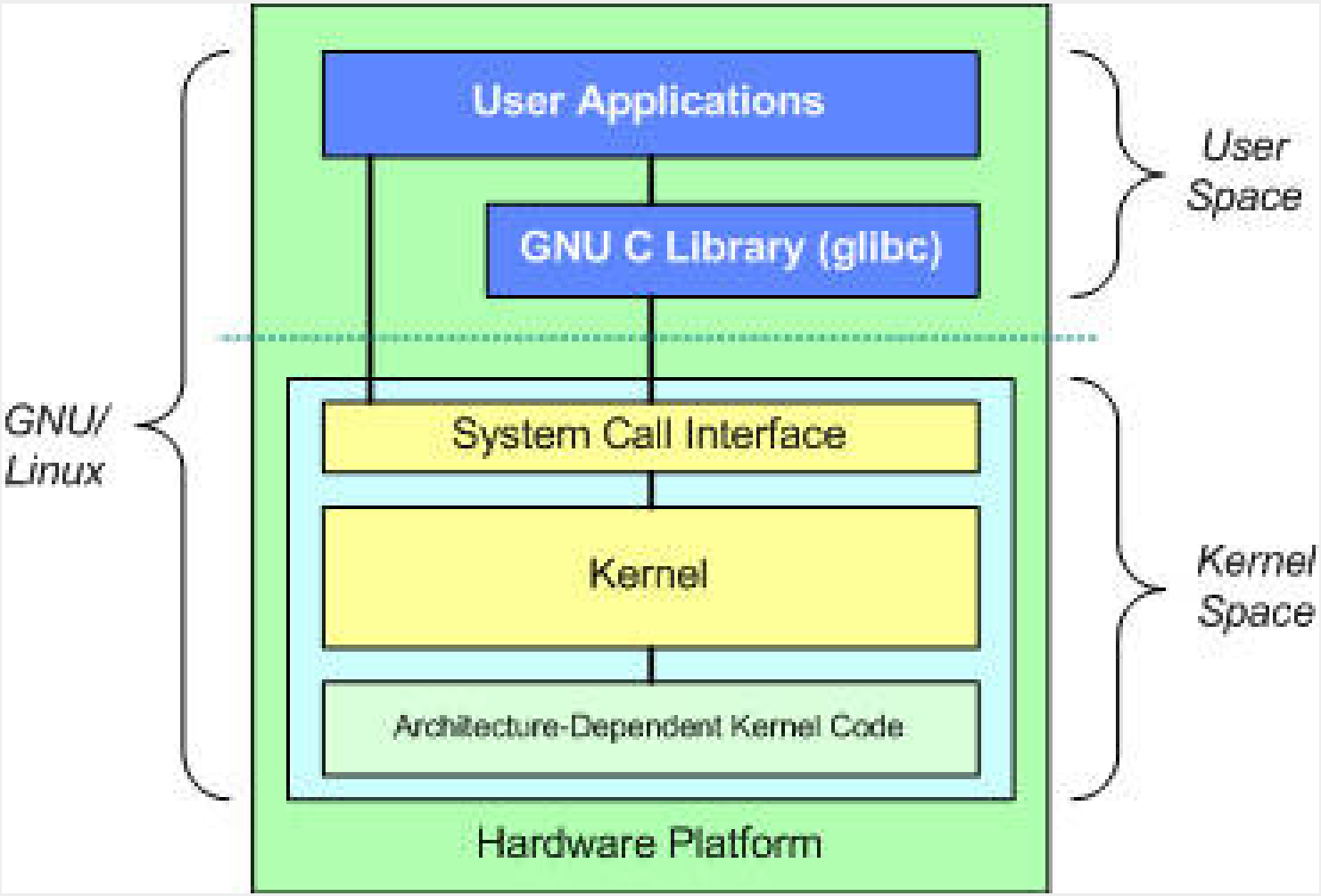
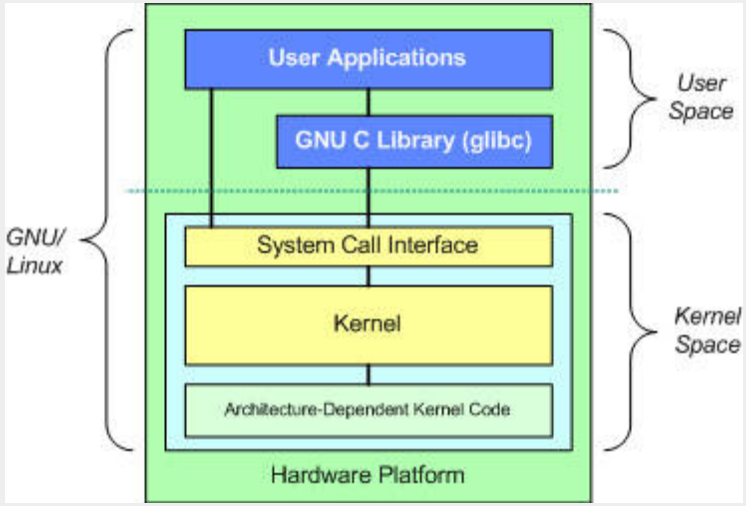


Figure: Architecture du système d'exploitation[1]

[1] https://developer.ibm.com/articles/l-linux-kernel/

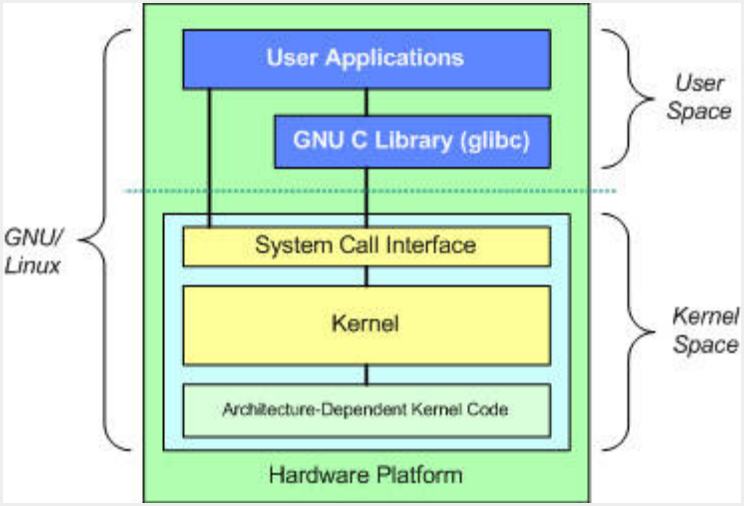
Architecture Linux

- Le noyau Linux est au cœur du système d'exploitation.
- Il permet aux logiciels d'interagir avec le matériel.
- Il fournit des services essentiels : gestion des processus, de la mémoire et des périphériques.



Architecture Linux

- Il existe une couche de traduction entre l'espace noyau et l'espace utilisateur. La traduction prend du temps.
- Les modules du noyau sont des extensions du noyau.



Vue d'ensemble du réseau Linux

Routage sous Linux

- Le noyau maintient les tables de routage.
- Les outils en espace utilisateur permettent d'interagir avec ces tables (ex. : commande `ip route`).

Firewall

- `iptables` : firewall en espace utilisateur.
- eBPF : machine virtuelle et firewall dans le noyau.

Flux des paquets

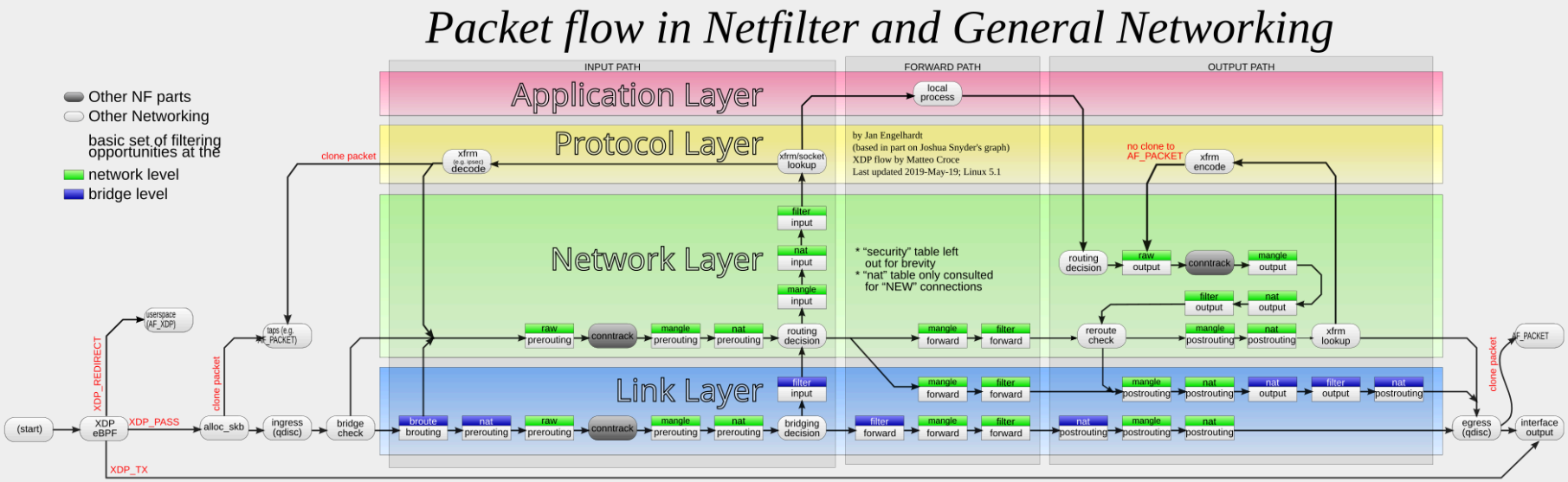


Figure: Flux des paquets Linux (Crédits : Jan Engelhardt)

iptables

`iptables` est un utilitaire en espace utilisateur permettant à un administrateur système de configurer le filtrage et le routage des paquets IP.

```
# Autoriser SSH
iptables -A INPUT -p tcp --dport 22 -j ACCEPT

# Bloquer tout autre trafic entrant
iptables -A INPUT -j DROP
```

eBPF

- eBPF est une machine virtuelle du noyau Linux : elle exécute des programmes personnalisés dans le noyau.
- Deux programmes : un pour le noyau et un pour l'espace utilisateur :
  - Application noyau écrite en C
  - SDK utilisateur disponible dans plusieurs langages
- Dérive du BPF, utilisé par exemple dans Wireshark.

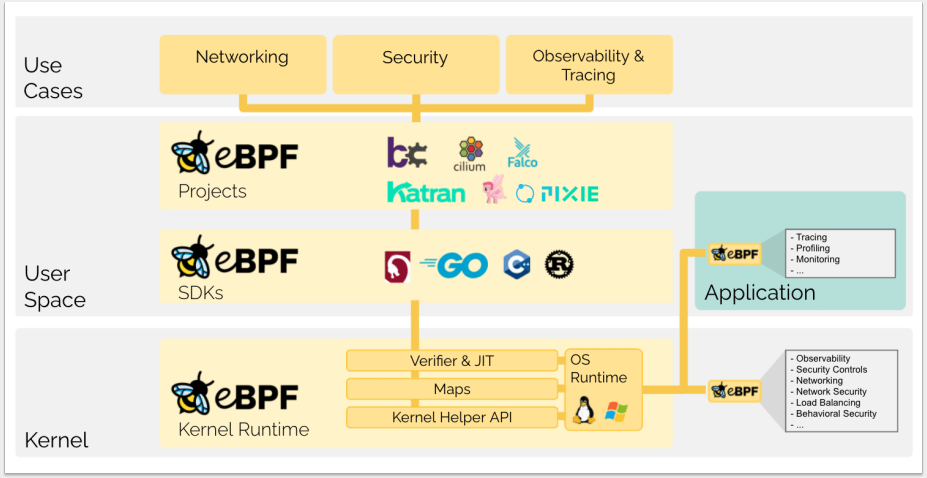


Figure: Architecture eBPF[2]

[2] https://ebpf.io/what-is-ebpf/

eBPF

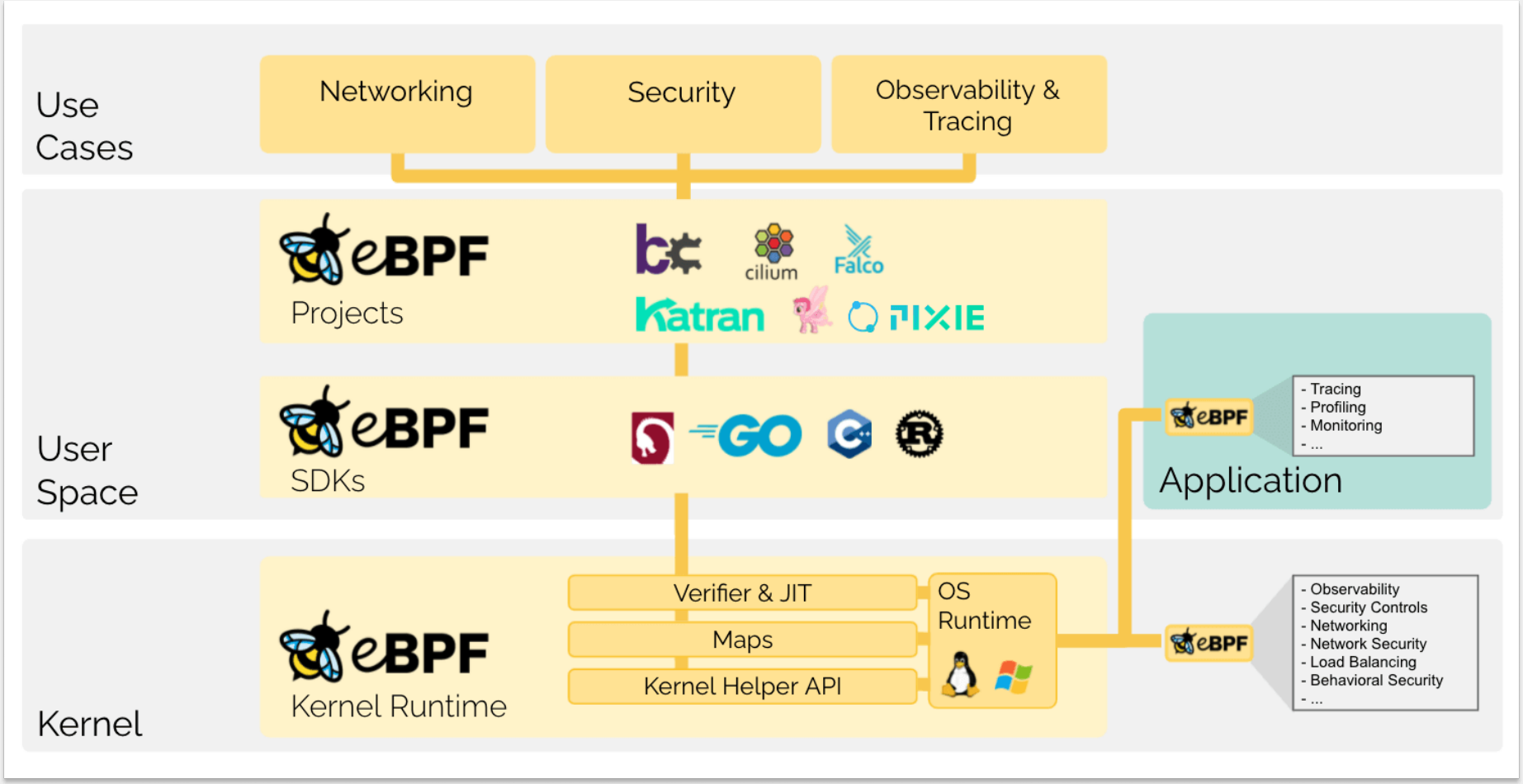


Figure: Architecture eBPF[3]

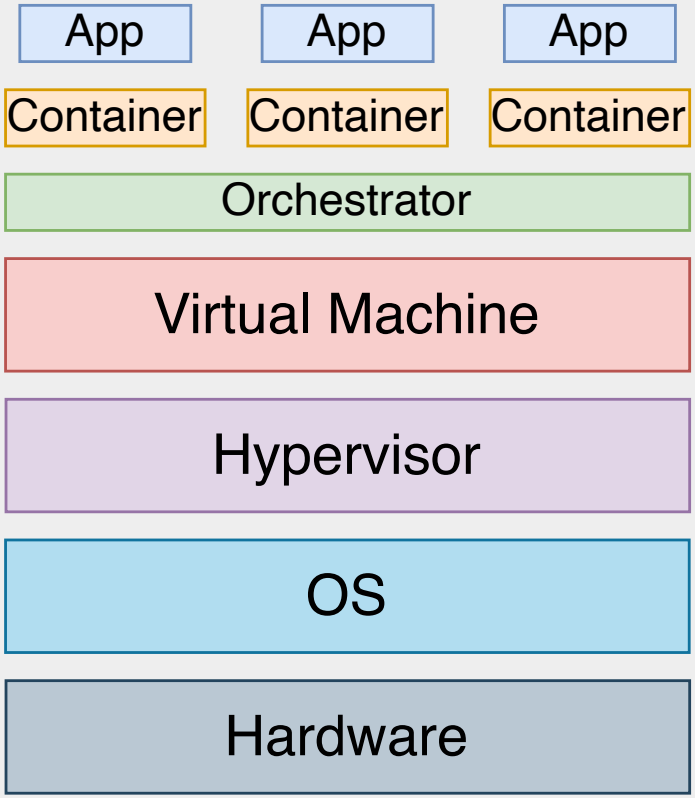
[3] https://ebpf.io/what-is-ebpf/

Virtualisation

Pourquoi la virtualisation

- Le matériel est coûteux
- Une seule machine peut être utilisée par plusieurs utilisateurs
- Réduction des coûts
- Moins de maintenance
- Réplication facilitée

Hyperviseurs



- Typiquement, la responsabilité d'un fournisseur de services cloud
  - Il existe (quelques) cas de virtualisation imbriquée

Le NIST (National Institute of Standards and Technology)[4]a défini une liste de fonctions et exigences de base.

[4] `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-125Ar1.pdf`

Fonctions de base d'un hyperviseur (1/2)

- **Isolation des processus de VM** — Planification et commutation de contexte entre les différents états du processeur lors de l'exécution d'applications dans les VM.
- **Médiation des périphériques et contrôle d'accès** — Fournit des périphériques aux VM et contrôle leur accès (ex. : carte réseau).
- **Exécution directe de commandes depuis les VM invitées** — Certaines commandes doivent être exécutées en contournant le système d'exploitation.

Fonctions de base d'un hyperviseur (2/2)

- **Gestion du cycle de vie des VM** — Création et gestion des images de VM, contrôle des états (Démarrer, Pause, Arrêt), migration, snapshots, supervision et application des politiques.
- **Gestion** — Comprend les mises à jour et les correctifs des modules logiciels.

Virtualisation

Hyperviseur

Deux types :

- Hyperviseurs **bare-metal** : communiquent directement avec le matériel via le noyau (ex. : KVM, ProxMox, Incus).
- Hyperviseurs **hébergés** : communiquent avec le système d'exploitation pour accéder au matériel (ex. : VirtualBox).

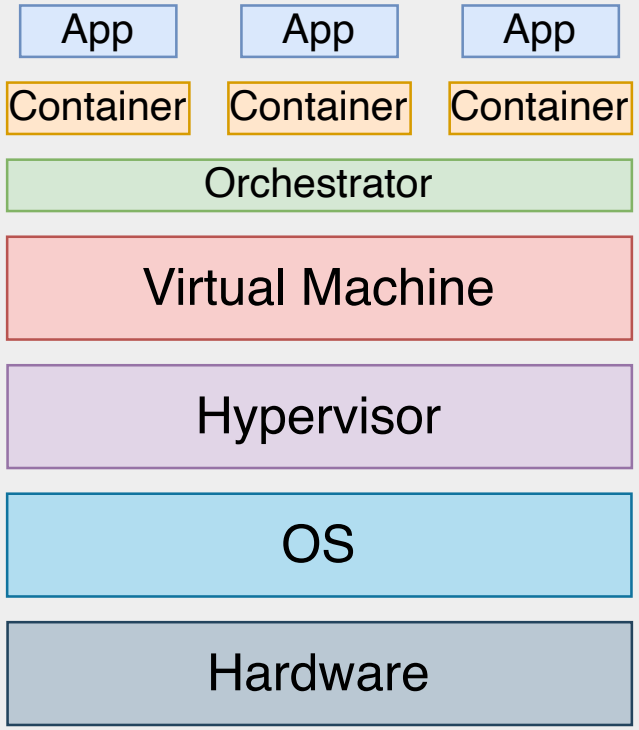


Figure: Pile de virtualisation

Pourquoi le bare-metal est meilleur

➜ Les processeurs modernes disposent d'instructions de virtualisation simplifiant la gestion des VM.
➜ Celles-ci permettent, par exemple, d'accéder directement à une partie de la mémoire ou à des périphériques (PCI passthrough).
➜ Les instructions s'exécutent directement sur le CPU.

Machine virtuelle (VM)

- Virtualisation complète du système d'exploitation
- Noyau individuel
- Peut avoir une architecture différente de celle de l'hôte

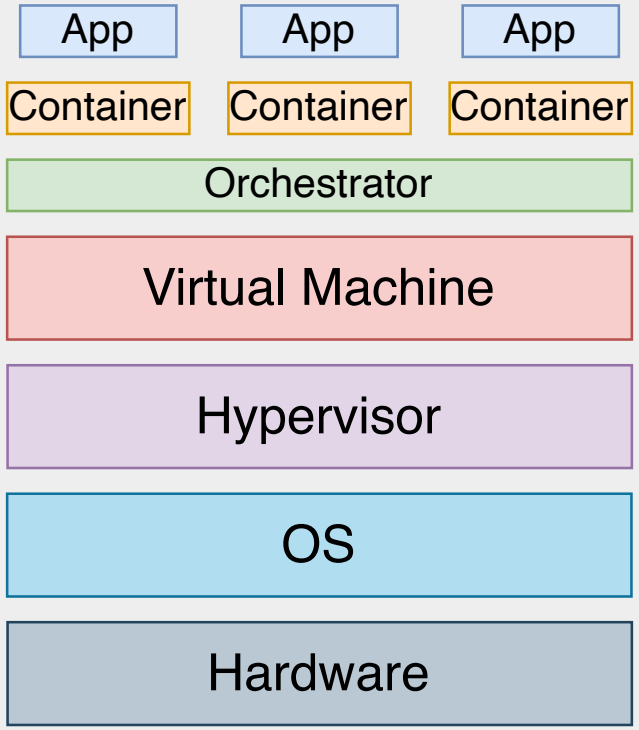


Figure: Virtualization Stack

Réseau virtuel

Linux dispose de capacités de réseau virtuel avancées utilisées pour héberger des VM et des conteneurs, ainsi que dans les environnements cloud. Elles servent notamment à :

- Permettre la communication entre VM et conteneurs
- Séparer le trafic
- Relier les conteneurs
- Créer des tunnels

Des VM aux conteneurs

Limites des machines virtuelles

1. Stockage : une VM occupe typiquement au moins 30 Go
   - Les données du noyau sont les mêmes si on utilise toujours Linux
   - Beaucoup de données sont dupliquées (ex. : bibliothèques)
2. Temps de démarrage : de l'ordre de la minute
3. Mutabilité : une VM garde son état, les retours arrière sont difficiles

Transition complexe vers les conteneurs

Virtualisation légère

Il y a toujours eu un besoin de virtualisation légère pour isoler efficacement les processus dans les systèmes multi-locataires.

Problèmes

- Manque de fonctionnalités dans le noyau pour isoler correctement les processus.
- Difficulté à *orchestrer* efficacement ces fonctionnalités.

Innovations clés du noyau Linux

- Permissions des processus
- Gestion des ressources
- Espaces de noms Linux (Namespaces)

Permissions des utilisateurs

- Traditionnellement, les utilisateurs et groupes ont des permissions sur les fichiers, dossiers et exécutables
  - `chmod g+rw file.txt`
- Soit avec sudo, soit sans
- Difficile d'avoir des permissions très fines

Problème : par défaut, un processus a les mêmes permissions que l'utilisateur

Permissions des processus : capacités Linux

Traditionnellement, un processus s'exécutait soit comme root, soit non. Les capacités Linux offrent un contrôle plus fin sur ce qu'un processus peut faire (utile pour la sécurité).

Exemples de capacités

- **CAP_NET_ADMIN** : autorise les opérations réseau (interfaces, tables de routage, etc.)
- **CAP_SYS_ADMIN** : le processus agit en tant qu'administrateur système
- **CAP_SYS_BOOT** : le processus peut redémarrer le système

Les capacités réduisent aussi le risque d'élévation de privilèges.

Gestion des ressources

Les *cgroups* (groupes de contrôle) sont une fonctionnalité du noyau Linux permettant d'allouer des ressources aux processus.

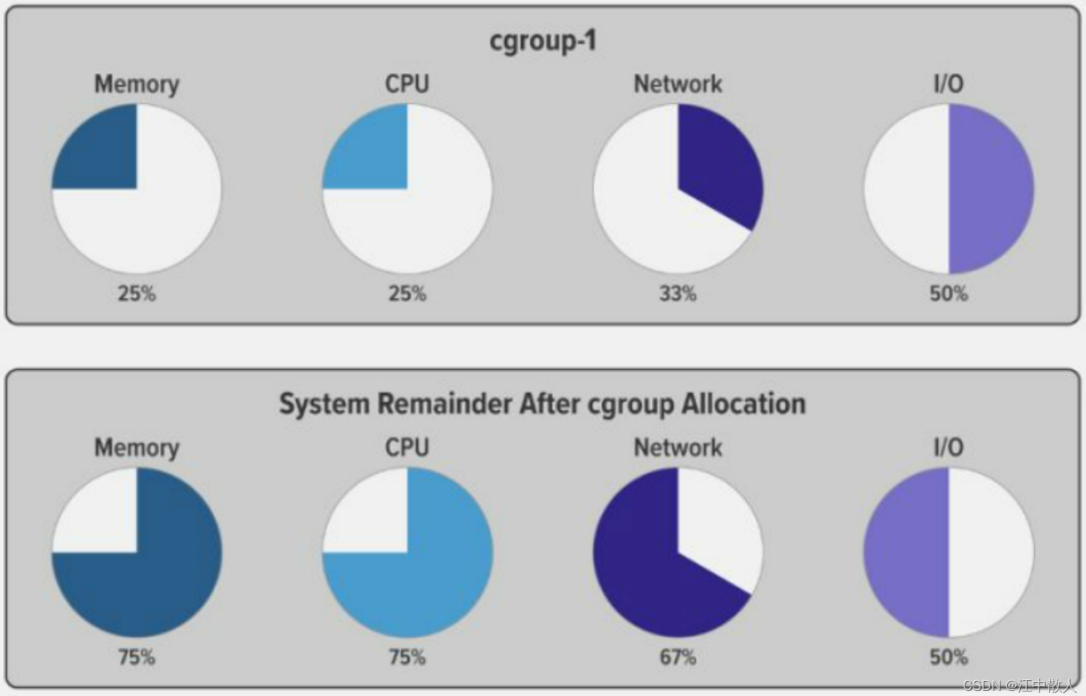


Figure: Cgroups

Espaces de noms Linux

Il est essentiel d'isoler les processus pour le multi-tenant et la sécurité.
Linux utilise le concept de namespace pour isoler différents éléments d'un processus tels que :

- Identifiants de processus (PID)
- Interfaces réseau
- Utilisateurs
- Points de montage
- Cgroups

Espaces de noms Linux – isolation des processus

```
1   struct nsproxy {
2   ...
3   struct uts_namespace     *uts_ns; # Utilisateur
4   struct ipc_namespace     *ipc_ns; # Processus
5   struct mnt_namespace     *mnt_ns; # Montage
6   struct pid_namespace     *pid_ns; # PID
7   struct net               *net_ns; # Reseau
8   struct time_namespace    *time_ns; # Temps
9   struct cgroup_namespace *cgroup_ns; # Ressources
0   };
```

Listing 1: Extrait du noyau Linux (`nsproxy.h`) (Lignes 32–42)

Conteneurs

Qu'est-ce qu'un conteneur ?

- Un conteneur est un processus léger comprenant un système de fichiers (Image) et un environnement d'exécution.
- Le système de fichiers du conteneur contient tout ce dont l'application a besoin :
  - Logiciels
  - Dépendances
  - Bibliothèques
  - Système de fichiers
  - Environnement

Les conteneurs partagent le noyau de l'hôte

Conteneurs Linux : caractéristiques

- Immutables : ne changent pas entre les exécutions
- Portables : faciles à partager et à déployer
- Scalables : plusieurs instances peuvent être exécutées simultanément
- Efficaces : démarrage rapide et faible consommation de ressources

Conteneurs Linux : résoudre les limites des VM

- **Isolation et efficacité**
  - Les namespaces Linux assurent l'isolation des processus, réseaux et systèmes de fichiers.
  - Les cgroups limitent et priorisent l'usage des ressources (CPU, mémoire, E/S disque. . .)
- **Portabilité et scalabilité**
  - Le système de fichiers en couches permet la mise en cache des données
  - Le *Copy on Write* facilite le déploiement simultané de plusieurs conteneurs
- **Immutabilité**
  - Chaque conteneur a une couche d'écriture au-dessus d'images en lecture seule, supprimée à la fin de l'exécution

Runtime de conteneur

Un *runtime de conteneur* est un logiciel qui aide à déployer et gérer les conteneurs durant tout leur cycle de vie.
Les runtimes de conteneurs ont favorisé leur adoption.

- Gestion des images
- Réseau
- Création et suppression des conteneurs
- Collecte des journaux
- Configuration de l'environnement

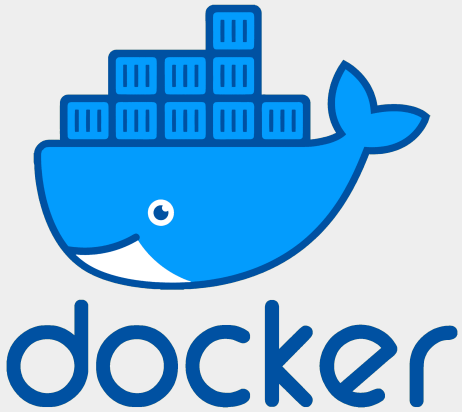


Figure: Docker : un runtime de conteneur populaire

Format d'image de conteneur (OCI)

En principe, il suffit d'un système de fichiers pour exécuter un conteneur.
L'**OCI** (Open Container Initiative) définit la façon d'emballer et de distribuer les images :

- Les images sont divisées en couches
- Une image OCI comprend un manifeste, une configuration et un ensemble de couches
- Le manifeste liste les couches et fournit les métadonnées de l'image
- La configuration décrit l'environnement d'exécution
- Les couches sont des archives tar contenant les modifications du système de fichiers

Containerfile

Exemple de Containerfile / Dockerfile

```
FROM alpine
COPY hello.sh /
ENTRYPOINT [ "/hello.sh" ]
```

Utilisation du Containerfile

- Série d'instructions pour automatiser la création d'une image
- (Presque) chaque instruction génère une nouvelle couche dans l'image

Liste complète des instructions :
`https://docs.docker.com/reference/dockerfile/`

Réseaux de conteneurs

Les conteneurs peuvent être reliés entre eux et à Internet via différents types de réseaux :

- **Bridge** : relie les conteneurs à l'hôte
- **Host** : connecte les conteneurs directement au réseau de l'hôte
- **Overlay** : relie des conteneurs sur plusieurs hôtes