Sécurité des systèmes d'exploitation

Conteneurs

Plan

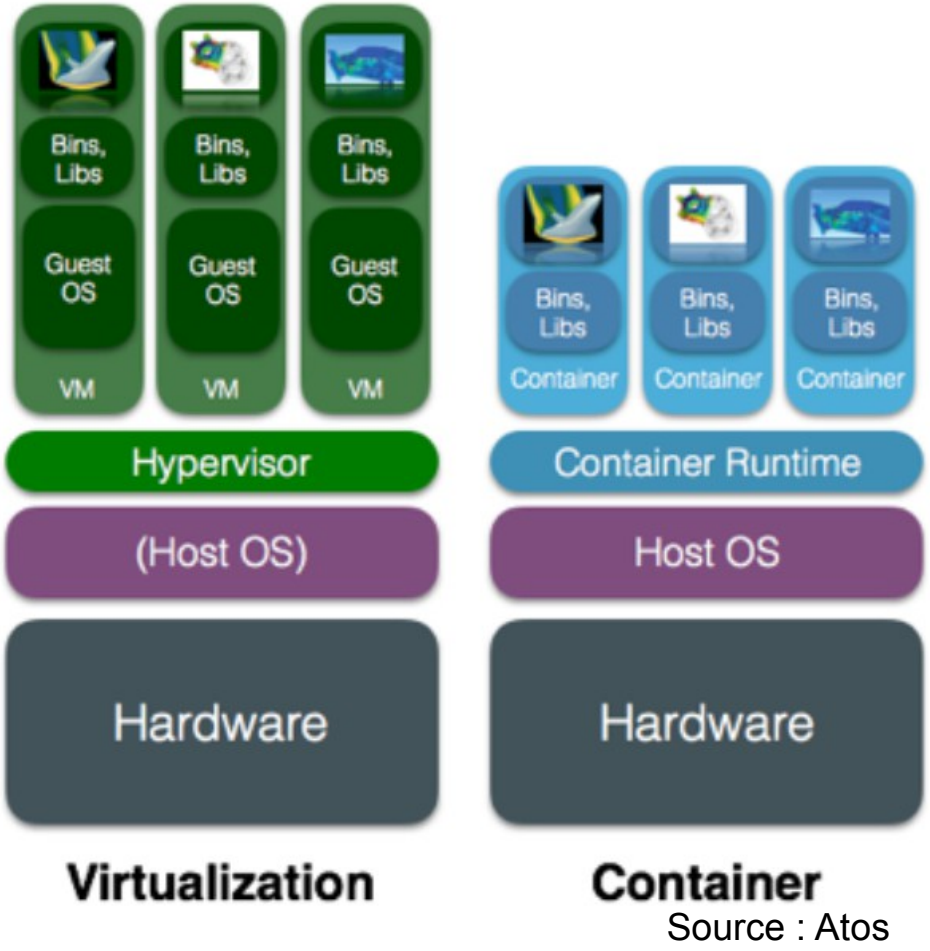
- Définition
- Mécanismes techniques
- Sécurité
- Implémentations courantes sous Linux
- Implémentation Windows

Définition

- Méthode pour faire fonctionner une application et ses dépendances (hors noyau) de façon autonome
  - Permet de distribuer facilement des applications
    - Image : contient l'application et ses dépendances (programmes, bibliothèques, fichiers de configuration, ...)
    - Conteneur : instance de l'application représentée par un ensemble cohérent de processus qui exécutent les programmes contenues dans l'image
    - Moteur de conteneur : logiciel en charge d'exécuter le contenu d'une image
      -

Définition

- Également appelée virtualisation légère ou *OS-Virtualization*
  - Ancêtres : chroot (1979), Linux VServer (2003)
  - Pas spécifique à Linux : FreeBSD *Jails* (2000), *Zones* Solaris (2005)
  - *Linux Containers* ou LXC (2008), Docker (2013)
- Reposent sur des mécanismes d'isolation offerts par le noyau



Source : Atos

Définition

- Un écosystème large s'est développé autour des technologies de conteneurs
  - Orchestrateur (par ex. Kubernetes)
  - Dépôts/registre d'image
- La sécurité de cet écosystème ne sera pas abordée ici
  - Voir Understanding Security Implications of Using Containers in the Cloud (Tak & al.) pour une introduction

Mécanismes techniques

- Conteneurs Linux reposent généralement sur
  - 4 composants principaux
    - Les cgroups (*Control Groups*) qui limitent les ressources matérielles (processeur, E/S disque, mémoire, ...)
    - Les espaces de noms (*namespaces*) qui présentent des « vues » différentes des ressources du système (réseau, processus, système de fichiers, utilisateurs, …)
    - L'appel système pivot_root() qui bascule dans l'environnement du conteneur en changeant la racine du système de fichiers
    - Les capacités Linux permettent de réduire les privilèges du compte root dans les conteneurs

Mécanismes techniques

- La sécurité des conteneurs repose sur
  - La bonne mise en œuvre des précédents composants
  - L'utilisation de mécanismes de sécurité existants :
    - Contrôle d'accès obligatoire (notamment AppArmor ou SELinux)
    - Filtrage des appels systèmes avec SECCOMP
  - La conception et le développement sécurisés du moteur de conteneur

Mécanismes techniques

- Cgroups (Control groups)
  - Partitionnement de l'ensemble des processus ou threads en des sous-ensembles (les « groupes ») sur lesquels peuvent s'appliquer différents contrôles, à vocation de performance ou de sécurité
    - Contrôles réalisés par des sous-systèmes du noyau appelé contrôleurs de ressources
    - V1 introduite dans le noyau 2.5.24, V2 dans le noyau 4.5
  - Configurables via /sys/fs/cgroup (pseudo-système de fichiers) ou avec des outils comme cgcreate, cgexec ou cgclassify
  - Chaque processus dispose d'une vue sur les groupes qui lui sont appliqués
    - /proc/self/cgroup

Mécanismes techniques

- Contrôleurs (V1) majeurs
  - CPU (cpu, cpuset, cpuacct) : utilisé pour restreindre un ensemble de processus à un nombre spécifique de processeur ou de temps processeur
  - *Memory* : contrôle l'allocation mémoire et ses limites d'un ensemble de processus
    - Les limites peuvent être figées ou souples
  - *Network* (net_cls) : permet d'étiqueter les paquets réseaux avec une valeur « classid », utilisable dans les règles iptables (filtrage de paquet) ou de la QoS
  - *Freezer* : mettre en pause (« geler ») un ensemble de processus via un signal SIGSTOP ou de les réveiller (« dégeler ») via un signal SIGCONT

Mécanismes techniques

- Contrôleurs (V1) majeurs (suite)
  - *Devices* : impose des restrictions sur l'accès aux périphériques à un ensemble de processus
    - Une approche en liste blanche est possible
      - Contenu courant de cette liste : /dev/null et /dev/urandom (mais pas /dev/random)
  - BLKIO : mise en place, pour un ensemble de processus, d'une politique d'accès (vitesse de lecture/écrite disque, opérations par seconde, contrôles des files, temps d'attente, …) aux périphériques de type bloc
  - PID : limitation du nombre de processus
    - Prévention des « fork-bombs » intentionnelles ou accidentelles

Mécanismes techniques

- Espaces de nom (*namespaces*)
  - Reposent sur un découpage logique
  - S'appliquent à un processus ou un groupe de processus (par ex., un conteneur)
  - 7 types différents

Espace de nom	Introduit dans le noyau
Mount	2.4.19
IPC	2.6.19
UTS	2.6.19
PID	2.6.24
Network	2.6.24
User	3.8
CGroup	4.6

Mécanismes techniques

- Espaces de nom (*namespaces*)
  - Créés à partir de l'appel système clone() en utilisant les drapeaux CLONE_NEW* lors de la création d'un processus
    - La création nécessite la capacité CAP_SYS_ADMIN (sauf l'espace de nom *user*)
    - 2 autres appels système existant pour manipuler les espaces de noms
      - setns() permet de rejoindre un espace de noms ciblés par un descripteur de fichier de type /proc/<pid>/ns/<namespace>
      - unshare() change le contexte d'exécution du processus appelant en le déplaçant, entre autre, vers un (ou plusieurs) nouvel espace de nom

Mécanismes techniques

- Types d'espaces de nom
  - *Mount* (CLONE_NEWNS) : permet d'avoir une vue spécifique des systèmes de fichiers montés
    - Le nouveau processus contient une copie des points de montage contenus l'espace de nom *Mount* de son parent
    - Monter et démonter des systèmes de fichiers n'affectera pas le reste du système (sauf pour les systèmes de fichiers montés partagés)
      - Permet de construire une topologie de systèmes de fichiers complètement différente puis de pivoter (via pivot_root()) vers une nouvelle racine et démonter l'ancienne
    - Peut aider à sécuriser indirectement d'autres espaces de noms en limitant l'accès à l'instance /proc de l'hôte (i.e. masquer les processus de l'hôte)

Mécanismes techniques

- Types d'espaces de nom (suite)
  - IPC (CLONE_NEWIPC) : concerne les IPC System V (par ex., les segments de mémoire partagée) et les files de messages POSIX
    - Ciblent des mécanismes IPC qui n'utilisent pas des chemins de fichiers pour désigner leurs objets
    - Isoler les communications interprocessus entre un conteneur et l'hôte (ou entre conteneurs)
  - UTS (CLONE_NEWUTS) : sépare les noms d'hôte et de domaine
    - Utile pour donner une « identité » différente à un conteneur (par ex., dans les journaux)

Mécanismes techniques

- Types d'espaces de nom (suite)
  - PID (CLONE_NEWPID) : permet de créer une nouvelle arborescence de processus commençant au PID 1
    - Les processus crées dans ce nouvel espace de nom ont un PID dans chaque espace de nom (cet espace de nom du processus et l'espace de nom « parent »)
      - Fonctionne de façon récursive en cas d'imbrication (profondeur limitée à 32 depuis le noyau 3.7)
    - Masque la visibilité des processus de l'espace de nom « parent » (par ex., les processus de l'hôte dans le cas d'un conteneur)
      - Par contre, les processus créés dans l'espace de nom enfant sont visibles depuis l'espace de nom parent

Mécanismes techniques

- Types d'espaces de nom (suite)
  - *Network* (CLONE_NEWNET) : créé un nouvel environnement réseau avec des instances différentes des interfaces réseau :
    - piles protocolaires IPv4 et IPv6, tables de routage IP, règles de pare-feu, arborescences /proc/net et /sys/class/net, sockets, …
    - une interface réseau physique ne peut être présente que dans un seul espace de noms
      - l'accès à cette interface depuis d'autres espaces de noms peut se faire via une interface virtuel
        - Permet de créer des tunnels entre des conteneurs ou un pont entre un conteneur et l'hôte

Mécanismes techniques

- Types d'espaces de nom (suite)
  - *User* (CLONE_NEWUSER) : permet de faire croire à un processus qu'il s'exécute en tant que root (uid 0) au sein de l'espace de nom alors qu'à l'extérieur de celui, l'identifiant est celui d'un utilisateur non-privilégié
    - Utilisation d'une table de correspondance entre les UID de l'hôte et du conteneur
    - Imbrication possible des espaces de nom *user* (comme pour PID)
      - profondeur limitée à 32 depuis le noyau 3.11

Mécanismes techniques

- Types d'espaces de nom (suite)
  - *User* (suite)
    - Sa création ne nécessite pas la capacité CAP_SYS_ADMIN
      - Le processus créé dispose de l'ensemble des capacités dans le nouvel espace de nom
        - Facilite la création de conteneur sans privilèges
    - L'exécution d'un binaire « setuid root », ou doté de toutes les capacités, au sein de l'espace de noms ne permet pas, par construction, d'utiliser ces privilèges à l'extérieur de l'espace de noms
      - N'exclue pas des défauts d'implémentation ...

Mécanismes techniques

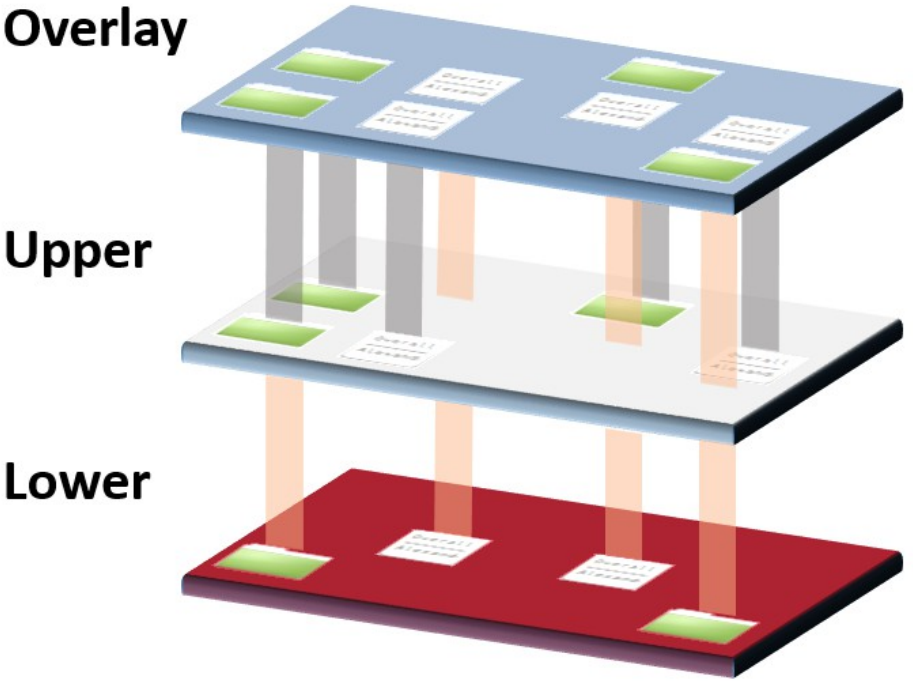
- Types d'espaces de nom (suite)
  - Cgroup (CLONE_NEWCGROUP) : créé, pour un processus, une vue (via /proc/self/cgroup) des Cgroups limitée à ceux qui lui sont appliqués
    - Les chemins de configuration des Cgroup de l'hôte sont masqués
      - Destiné à prévenir des fuites d'informations de l'hôte qui seraient visibles depuis le conteneur (par ex., sur le moteur de conteneur utilisé)
      - Facilite la migration de conteneurs entre machines

Mécanismes techniques

- Commencée depuis la version 2.4 du noyau, l'évolution des espaces de noms n'est pas terminée
  - Intégration progressive car ne fait pas partie de la conception initiale => source de bugs et de vulnérabilités
  - Propositions de nouveaux espaces de nom : syslog, périphériques, le temps, les pseudo-systèmes de fichiers proc et sys, ...
  - Cas particulier des LSM « lourds »
    - L'hôte fournit AppArmor, un conteneur utilise SELinux pour sa sécurité => peut-on faire cohabiter les 2 ?
      - Pas possible avec un noyau Linux récent (4.17)
        - Espace de noms dédié à l'avenir ?

Mécanismes techniques

- Utilisation de systèmes de fichiers en couche
  - Partage d'une base commune issue de l'hôte
  - Seules les modifications sont enregistrées (via *Copy on Write*) sur la couche la plus haute
  - AUFS (dérivé de UnionFS), OverlayFS



Source : Datalight

Sécurité

- Les risques liés aux conteneurs sont très liés au noyau Linux
  - Maturité de l'implémentation des espaces de noms
  - Gestion des capacités et des privilèges du compte root
  - Noyau partagé par tous les conteneurs
    - Présente généralement une surface d'attaque importante
    - Maîtrise des vulnérabilités du noyau ?
- Solution 100 % logicielle (pas de recours à des mécanismes d'isolation apportés par le matériel)
  - Gain en flexibilité mais risque plus élevé si des périphériques sont directement exposés

Sécurité

- Menaces applicables
  - Évasion du conteneur vers l'hôte, reposant sur :
    - Absence (ou mauvaise configuration) d'espaces de noms
    - Configuration par défaut non-sécurisée
      - Persistance de capacités non-indispensables
    - Exposition de l'hôte via procfs et sysfs (obtention d'informations utiles pour un attaquant ou reconfiguration de l'hôte en cas de mauvaise protection), par ex. :
      - Modification des variables noyau /proc/sys/ via sysctl()
      - Configuration du noyau (via /proc/config.gz si le paramètre CONFIG_IKCONFIG_PROC a été activé)
      - /proc/sys/security (securityfs) : accès à la configuration de LSM comme AppArmor ou SELinux
      - /sys/firmware/efi/vars : interaction avec les variables (U)EFI
    - Interface réseau partagée avec l'hôte

Sécurité

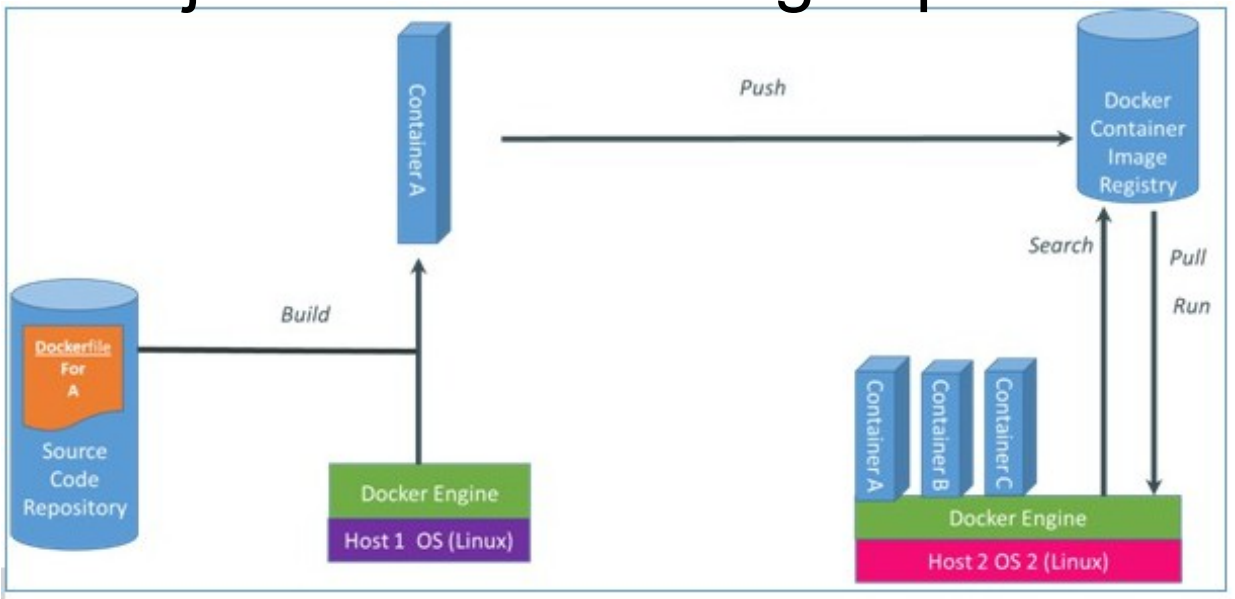
- Menaces applicables (suite)
  - Attaques entre conteneurs
  - Attaques ciblant le conteneur
    - Exposition du conteneur à des réseaux potentiellement hostiles
    - Utilisation d'images avec une base logicielle élargie
      - Présence d'outils facilitant les actions de l'attaquant (par ex., interpréteur Python) ou son accès distant (par ex., ssh ou netcat)
    - Déni de service et consommation excessive de ressources
      - Processeur, mémoire, E/S stockage
      - Génération d'aléa : un conteneur peut assécher les réservoirs d'entropies du noyau via /dev/random ou l'appel système getrandom()

Sécurité

- Menaces applicables (suite)
  - Attaques sur le moteur de conteneur
    - Plusieurs attaques ont été recensées sur Docker et LXC :
      - L'accès à procfs (mauvaise gestion du montage)
      - Gestion des liens (symbolique ou physique)
      - Attaque de type *Directory traversal*
      - Fichiers sensibles accessibles en écriture
- Menaces non-spécifiques
  - Attaques avancées ciblant du matériel
  - Utilisation de code non éprouvé dans le conteneur
    - Logiciels non-mis à jour ou développés sans prise en compte de la sécurité
      - Le conteneur n'apporte aucune protection contre les mauvaises pratiques ...

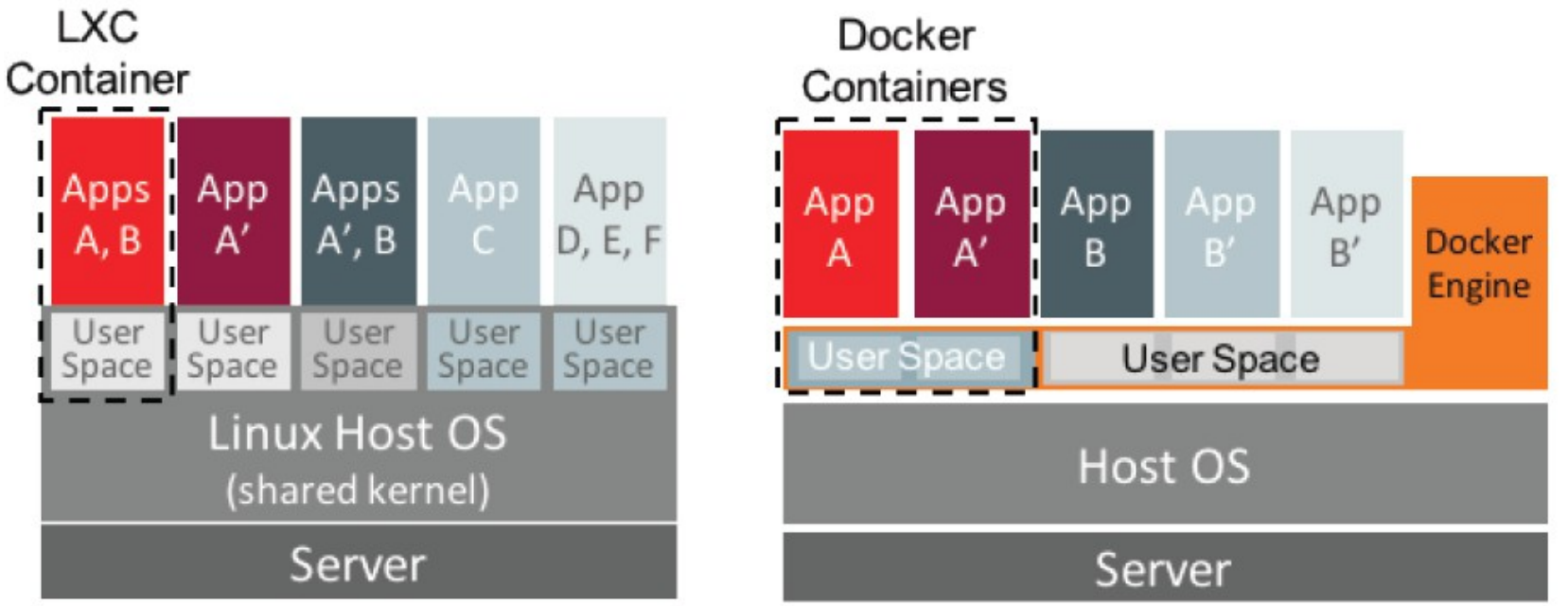
Sécurité

- Attaques indirectes
  - Concentration du code des outils sur des dépôts comme GitHub
    - Réaction en cas de compromission de GitHub ?
  - Images malveillantes ou compromises
    - Contrôle du contenu, mécanismes d'attestation ?
  - Gestions des mises à jour dans des images produites par des tiers



Implémentations courantes sous Linux

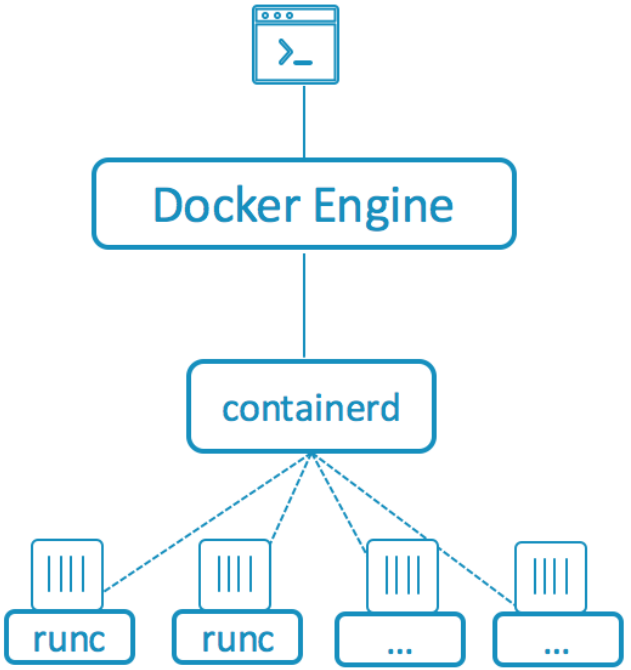
- Deux approches différentes
  - LXC : « conteneurisation » de système d'exploitation
    - Chaque conteneur a son propre espace utilisateur isolé et démarre depuis un « vrai » processus init
  - Docker : « conteneurisation » d'application



Source : Oracle

Implémentations courantes sous Linux

- Docker
  - Solution de référence en 2018
    - Fonctionne aussi sous MacOS et Windows
  - basée sur *runC* et *containerd* depuis la version 1.11



Same Docker UI and commands

User interacts with the Docker Engine

Engine communicates with containerd

containerd spins up runc or other OCI compliant runtime to run containers

Source : Docker Inc

Implémentations courantes sous Linux

- Docker (suite)
  - runC
    - Outil CLI pour créer des conteneurs et les exécuter
    - Implémente la spécification OCI (Open Container Initiative)
    - Construit sur Libcontainer (apporte la gestion de primitives du systèmes comme les espaces de nom, les *cgroups*, les capacités, ...)
  - Containerd - *daemon* pour contrôler runC
    - Permet de gérer le cycle de vie d'un conteneur (transfert et stockage de l'image, exécution et supervision du conteneur, ...)
    - Conçu pour être intégré dans des grands systèmes, ne cible pas directement les utilisateurs finaux

Mécanismes techniques

- Dockerfile
  - Fichier texte contenant des directives de configuration (plus ou moins nombreuses) qui instrumentent la génération de l'image et du conteneur résultant
  - Permet notamment de dériver des images existantes pour les adapter à un besoin particulier

```
FROM openjdk:8-jdk-stretch

RUN apt-get update && apt-get install -y git curl && rm -rf /var/lib/apt/lists/*

ARG user=jenkins
ARG group=jenkins
ARG uid=1000
ARG gid=1000
ARG http_port=8080
ARG agent_port=50000
ARG JENKINS_HOME=/var/jenkins_home

ENV JENKINS_HOME $JENKINS_HOME
ENV JENKINS_SLAVE_AGENT_PORT ${agent_port}

# Jenkins is run with user `jenkins`, uid = 1000
# If you bind mount a volume from the host or a data container,
# ensure you use the same uid
RUN mkdir -p $JENKINS_HOME \
  && chown ${uid}:${gid} $JENKINS_HOME \
  && groupadd -g ${gid} ${group} \
  && useradd -d "$JENKINS_HOME" -u ${uid} -g ${gid} -m -s /bin/bash ${user}

# Jenkins home directory is a volume, so configuration and build history
# can be persisted and survive image upgrades
VOLUME $JENKINS_HOME

# `/usr/share/jenkins/ref/` contains all reference configuration we want
# to set on a fresh new installation. Use it to bundle additional plugins
# or config file with your custom jenkins Docker image.
RUN mkdir -p /usr/share/jenkins/ref/init.groovy.d

# Use tini as subreaper in Docker container to adopt zombie processes
ARG TINI_VERSION=v0.16.1
COPY tini_pub.gpg ${JENKINS_HOME}/tini_pub.gpg
RUN curl -fsSL https://github.com/krallin/tini/releases/download/${TINI_VERSION}/tini-static-$(dpkg --print-architecture) -o /sbin/tini \
  && curl -fsSL https://github.com/krallin/tini/releases/download/${TINI_VERSION}/tini-static-$(dpkg --print-architecture).asc -o /sbin/tini.asc \
  && gpg --no-tty --import ${JENKINS_HOME}/tini_pub.gpg \
  && gpg --verify /sbin/tini.asc \
  && rm -rf /sbin/tini.asc /root/.gnupg \
  && chmod +x /sbin/tini

COPY init.groovy /usr/share/jenkins/ref/init.groovy.d/tcp-slave-agent-port.groovy

# jenkins version being bundled in this docker image
ARG JENKINS_VERSION
ENV JENKINS_VERSION ${JENKINS_VERSION:-2.121.1}

# jenkins.war checksum, download will be validated using it
ARG JENKINS_SHA=5bb075b81a3929ceada4e960049e37df5f15a1e3cfc9dc24d749858e70b48919

# Can be used to customize where jenkins.war get downloaded from
ARG JENKINS_URL=https://repo.jenkins-ci.org/public/org/jenkins-ci/main/jenkins-war/${JENKINS_VERSION}/jenkins-war-${JENKINS_VERSION}.war

# could use ADD but this one does not check Last-Modified header neither does it allow to control checksum
# see https://github.com/docker/docker/issues/8331
RUN curl -fsSL ${JENKINS_URL} -o /usr/share/jenkins/jenkins.war \
  && echo "${JENKINS_SHA}  /usr/share/jenkins/jenkins.war" | sha256sum -c -

ENV JENKINS_UC https://updates.jenkins.io
ENV JENKINS_UC_EXPERIMENTAL=https://updates.jenkins.io/experimental
ENV JENKINS_INCREMENTALS_REPO_MIRROR=https://repo.jenkins-ci.org/incrementals
RUN chown -R ${user} "$JENKINS_HOME" /usr/share/jenkins/ref

# for main web interface:
EXPOSE ${http_port}

# will be used by attached slave agents:
EXPOSE ${agent_port}

ENV COPY_REFERENCE_FILE_LOG $JENKINS_HOME/copy_reference_file.log

USER ${user}

COPY jenkins-support /usr/local/bin/jenkins-support
COPY jenkins.sh /usr/local/bin/jenkins.sh
COPY tini-shim.sh /bin/tini
ENTRYPOINT ["/sbin/tini", "--", "/usr/local/bin/jenkins.sh"]

# from a derived Dockerfile, can use `RUN plugins.sh active.txt` to setup /usr/share/jenkins/ref/plugins from a support bundle
COPY plugins.sh /usr/local/bin/plugins.sh
COPY install-plugins.sh /usr/local/bin/install-plugins.sh
```

```
# Alpine Linux with OpenJDK JRE
FROM openjdk:8-jre-alpine
# copy WAR into image
COPY spring-boot-app-0.0.1-SNAPSHOT.war /app.war
# run application with this command line
CMD ["/usr/bin/java", "-jar", "-Dspring.profiles.active=default", "/app.war"]
```

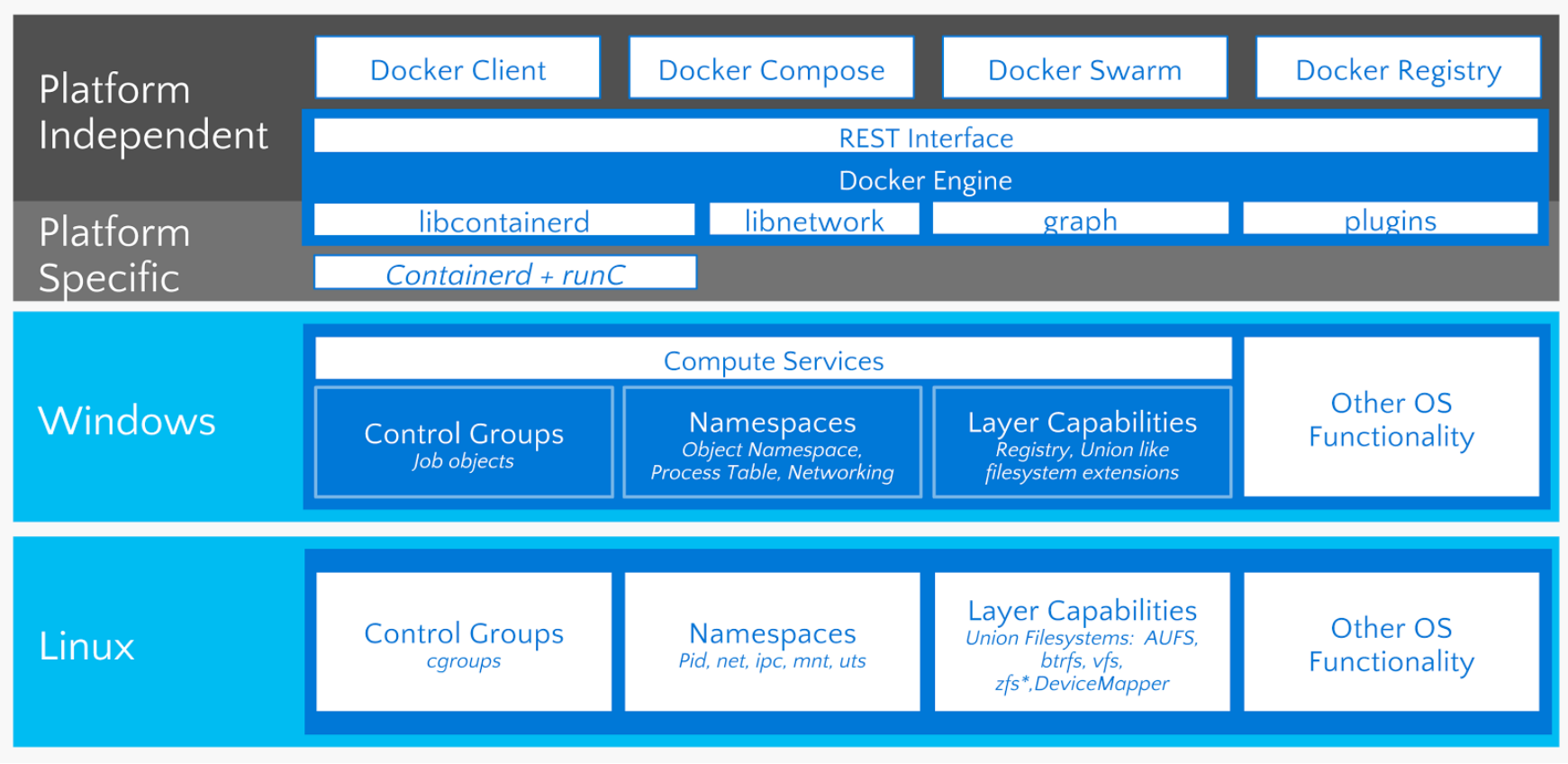
Source : Stackify

Implémentations courantes sous Linux

- Linux Containers (LXC)
  - Reprend des concepts initiés avec Linux VServers
  - Utilise les Cgroups et les espaces de nom
    - Ainsi que les profils AppArmor et SELinux, le filtrage SECCOMP, les capacités
  - Conteneurs privilégiés (uid 0 du conteneur => uid 0 de l'hôte) / non-privilégiés (uid 0 du conteneur = utilisateur non privilégié de l'hôte)
- LXD
  - Gestionnaire de conteneur construit au dessus de LXC
    - Apporte le support des dépôts d'images et des options comme l'accès direct aux périphériques (GPU, USB, ...)

Implémentation Windows

- Intégration récente des conteneurs
  - Depuis Windows 10 1607 & Windows Server 2016
  - Portage du moteur Docker
    - *Compute Service* remplace runC et containerd



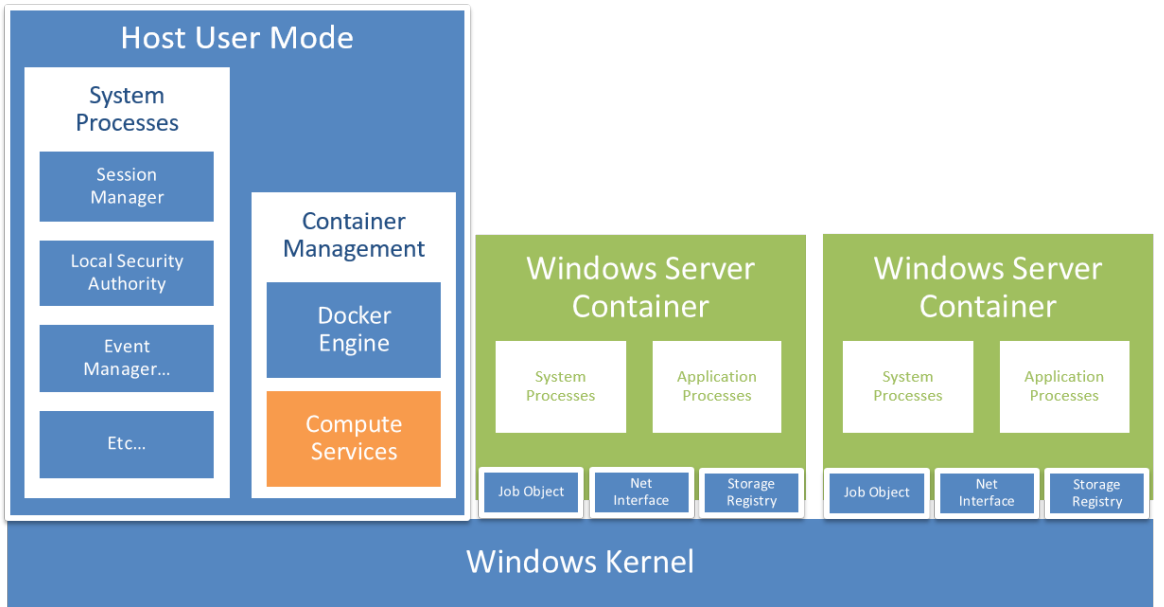
Source : Microsoft

Implémentation Windows

- Docker est construit sur des mécanismes inhérents au noyau Linux (*namespace*, *Cgroup, ...*)
  - Adaptation à Windows
    - Introduction d'un nouvel objet Silo (extension des objets de type Job) permettant de regrouper des processus et contrôler les ressources
    - L'objet Silo permet également de virtualiser l'espace de nom des objets noyau, utile pour séparer différentes instances de certains objets
      - L'espace de nom des objets (vus du gestionnaire d'objet)
      - Registre
      - Système de fichiers
      - Session
      - Réseau

Implémentation Windows

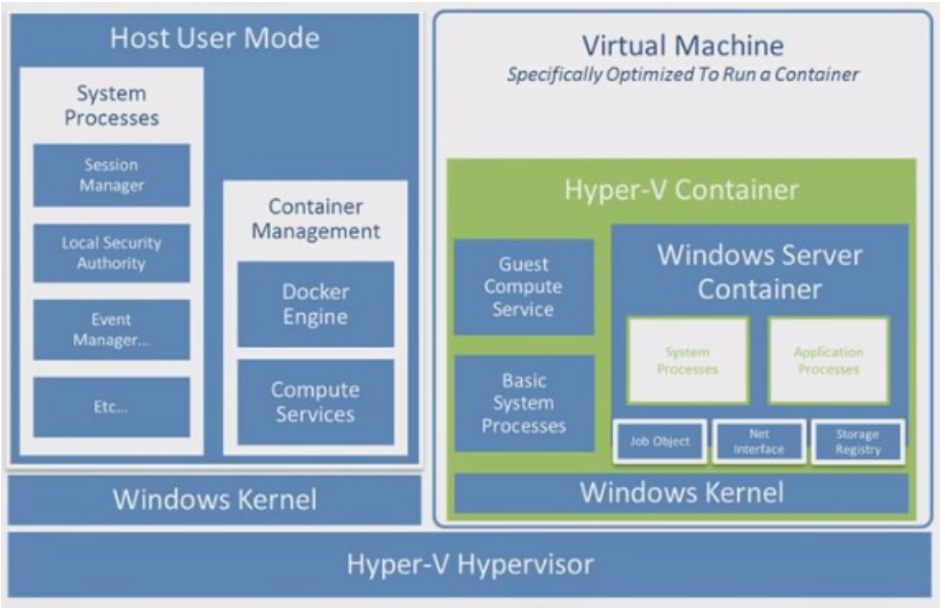
- 2 approches :
  - *Windows Server Containers*
    - Utilisation des nouveaux mécanismes du noyau
      - Sécurité ?
        - *These containers do not provide a hostile security boundary and should not be used to isolate untrusted code.* Microsoft
    - Ne fonctionne qu'avec des images construit sous Windows (noyau partagé entre l'hôte et les conteneurs)



Source : Microsoft

Implémentation Windows

- 2 approches (suite)
  - *Hyper-V Containers*
    - Un conteneur = une VM
      - Approche présentant une meilleure isolation, se retrouve sous Linux avec Intel Clear Linux (intégration en cours dans Kata Containers)
    - Utilisation de VM Linux pour héberger des images construit sous Linux



Source : Microsoft

Pour approfondir

- Namespaces in operation, Michael Kerrisk, 2013-2016
- Understanding and Hardening Linux Containers, Aaron Grattafiori (NCC Group), 2016
- Abusing Privileged and Unprivileged Linux Containers, Jesse Hertz (NCC Group), 2016
- Docker security, Docker inc

Questions ?