Veille technologique sur les runtimes

I. Les runtimes de containers choisis et les motifs derrière leur choix

i. runc

Quand Docker exécute une image, runc est le composant qui s'occupe de l'exécution de l'image par défaut. L'image exécutée suit la norme OCI (Open Container Initiative), donc une exécution d'images plus facile sans arguments additionels. En plus, il est considéré le runtime le plus compatible avec Docker, car runc est écrit en Go, en plus il a une performance native car il ne dépend pas sur les boot VM ni l'interception de syscalls, car sa sécurité est réalisée avec seccomp et le choix des capabilités Linux. [1]

ii. Youki

L'image exécutée suit la norme OCI (Open Container Initiative), En plus le runtime est développé en Rust, ainsi utilisation sécurisée de la mémoire avec moins d'empreintes et maintenance continue. [2]

iii. crun

Quand Podman exécute une image, crun est le composant qui s'occupe de l'exécution de l'image par défaut. Caractérisé par un temps de démarrage d'image 5 fois plus rapide que runc, et plus de performance. La principale raison vient de la langue de programmation utilisée pour le développement, C en étant compatible OCI [2] (dans le tableau des métriques mesurés)

iv. styrolite [3]

Il assure des fonctionalités de sécurité en plus à l'image exécutée comme: offrir des sandboxs programmables et contrôler les appels systèmes à travers un hypervisor de type 1. En plus il est développé en Rust, menant à moins d'empreintes mémoires. [4]

vi. les projets abandonnés

- rkt : un runtime de conteneurs pod-native pour Linux, donc il implémente la fonctionalité des pods, qui est un composant entrant dans un environnement scalable d'exécution de plusieurs containers, mais avec moins d'overhead que la technologie d'orchestration, Kubernetes. En effet, le projet a terminé il y a 5 ans. Les conférences de CNCF m'ont permis de connaître le runtime qui a connu un grand succès à l'époque. [5]
- nabla : Un runtime se basant sur l'émulation à travers la redirection des appels systèmes vers une LibOS (bibliothèque système utilisateur), ce qui permet de garantir la sécurité sans recourir aux techniques de virtualisation de microVMs, qui sont considérés plus lentes à cause de leur overhead. Son support est terminé il y a 5 ans. [6]

II. Comparaison de privilèges

	runc	Youki	crun	styrolite
Mode privilégié	Oui, en ajoutant toutes les capabilités au fichier config.json généré par runc spec	Oui, en ajoutant toutes les capabilités dans le fichier config.json généré par youki spec	Oui, en ajoutant toutes les capabilités dans le fichier config.json généré par crun spec	Oui, en ajoutant les capabilités au fichier config.json, puis exécuter l'image avec le nouveau config.json
Isolation par namespace	pid, network, ipc, cgroup, mount, uts dans le fichier config.json généré par runc spec	pid, network, ipc, uts, mount, cgroup dans le fichier config.json généré par youki spec	pid, network, user, ipc, cgroup, mount, uts dans le fichier config.json généré par crun spec	pid, network, ipc, cgroup, mount, uts, user, time dans le fichier namespace.rs repo Github

Support de seccomp	Oui, la rubrique “Build Tags” libseecomp [7]	Oui, dans la rubrique Dependencies il intègre libseccomp [2]	Oui , un fichier dédié aux communications avec seccomp [8]	Non, le runtime assure la sécurité en exécutant les images dans des zones d’exécution isolées comme des machines virtuelles légères, il n’assure pas la sécurité par seccomp. [9]
AppArmor / SELinux	Non, la rubrique “Build Tags”, mais SElinux et AppArmor peuvent changer de contexte et exécuter une image runc [7]	Oui [10], [11]	Oui, le fichier linux.h dans le repo a des fontions pour ajouter les privilèges aux profiles selinux et apparmor (e.g libcrun_set_selinux label)	Non en cherchant par mot clé sur le repo Github, mais SElinux et AppArmor peuvent changer de contexte et exécuter une image styrolite [12]
Filesystem en lecutre seulement	Oui, à la ligne readonly: true dans le fichier config.json généré par runc spec	Oui, à la ligne readonly: true dans le fichier config.json généré par youki spec	Oui, à la ligne readonly: true dans le fichier config.json généré par crun spec	Oui [13]
Mode rootless	Oui, à condition que le user namespaces soit activé et compilé dans le kernel [14]	Oui, dans la rubrique Rootless Container [2]	Oui, avec crun –rootless	Non, il est possible de modifier les mappings dans le fichier config.json mais il n’est pas possible d’exécuter en mode rootless

Les privilèges

- Mode privilégé : il permet d’avoir toutes les capabilités du système hôte, ainsi avoir tout le contrôle sur le système hôte. N.B: Le fichier config.json c’est le fichier généré par un runtime qui décrit les privilèges d’un containeur exécuté.
- Support seccomp : un appel système qui filtre les syscalls ainsi contribuant à moins de droits d’utilisateur de communication avec le kernel [15]
- AppArmor / SELinux: définir les ressources ou définir les privilèges d’un container. [16]
- Filesystem en readonly: éviter la modification du filesystem par un utilisateur non-privilégé. [17]
- Mode rootless: Le remappage pour le user namespace afin d’exécuter les containeurs en étant non-root ce qui limite à l’utilisateur l’accès à certaines capabilités, limitant l’accès au filesystème. [18]

Capabilités [19]

	runc mode non-privilégié	youki mode non-privilégié	crun mode non-privilégié	styrolite mode non-privilégié
SYS_ADMIN	Non	Non	Non	Non
NET_ADMIN	Non	Non	Non	Non
SYS_PTRACE	Non	Non	Non	Non

SYS_MODULE	Non	Non	Non	Non
SYS_TIME	Non	Non	Non	Non
AUDIT_WRITE	Oui / P-E-B	Oui	Oui / P-E-B-A	Oui / P-E-B-A-I
NET_BIND_SERVICE	Oui / P-E-B	Oui	Oui / P-E-B-A	Oui / P-E-B-A-I
KILL	Oui / P-E-B	Oui	Oui / P-E-B-A	Oui / P-E-B-A-I

Les capabilités étaient récupérés respectivement par:

- runc spec
- youki spec
- crun spec
- par un des trois fichiers config.json générés par l'un des 3 commandes précédentes, car styrolite n'a pas une commande de génération d'un fichier config.json.

Les types de capabilités

Permitted (P): Les capabilités sont automatiquement permises au thread sans regarder les capabilités du master thread.

Effectifs (E): les capabilités utilisés dans les checks de permission du kernel.

Bounding (B): les capabilités utilisés durant l'appel system execve2 appelé avec le chemin de la commande.

Ambient (A): Les capabilités restent activé avec le processus appelé sans avoir besoin d'être privilégié.

Inheritable (I): Les capabilités sont gardés au runtime lors de l'appel de execve(2)

– En premier lieu, les capabilités qui doivent être desactivés par défault

i. CAP_SYS_ADMIN : ELle offre un nombre de privilèges d'administration système tel que l'accès aux appels systems (mount, unmount, pivot_root), de surpasser le filesystem, créer de nouveaux namespaces avec clone(2), et bien d'autres dans la page manuel de Linux.

ii. CAP_NET_ADMIN: qui permet de configurer les adresses IP , les tables de routages , il doit activé dans les containers de réseaux pas dans les environnements partagés.

iii. CAP_SYS_PTRACE: Avec cette capabilité , un processus peut lire la mémoire d'un autre processus pour déboggage, il doit être activé dans les environements de déboggage sécurisés, sinon trop risqué.

iv. CAP_SYS_MODULE: Charger des modules kernels, peut entraîner à des vulnérabilités dans le containeur pour entrer en mode kernel privilégié.

v. CAP_SYS_TIME : Changer le temps système et affecter le résultat des logs.

– En deuxième lieu, les capabilités qui sont activés par défault.

ii. CAP_AUDIT_WRITE: Écrire dans les infos d'audit existants dans le kernel.

iii. CAP_NET_BIND_SERVICE: Permet l'accès à numéro port plus petit que 1024 même si l'utilisateur n'a pas les privilèges de root.

iv. CAP_KILL: Envoyer des signaux aux processus comme SIG_KILL , l'utilisateur ne peut pas tuer d'autres process que les siens.

III. Limites

- Dépendance de la sécurité du kernel de l'hôte

Les containeurs héritent les vulnérabilités du kernel (e.g., Dirty Pipe, qui permet d'écraser des données selon des fichiers arbitraires en lecture seulement [20] les dépassements de seccomp qui filtre les appels système privilégiés.).

- La mémoire utilisée par défaut par tmpfs n'est pas limitée

tmpfs est un système de fichiers de linux, permettant de stocker temporairement les fichiers à l'extérieur de la couche sur lequel le containeur exécute. À l'arrêt du container l'image est arrêtée. Il est plus rapide que d'autres sytème de fichiers, notamment overlayFS [21] en montage de filesystème. Mais dans certains runtimes tmpfs est configuré sans limite par défaut sans limite, ce qui peut mener à des craches OOM (Out of memory), [22] et tmpfs peut être configurée par un utilisateur malicieux pour mener à des attaques DDoS (Denial of Service attack) [23] La limite peut être vue dans /dev/shm

- Incompatibilité avec Macvlan et ipvlan

Macvlan est un driver Linux de réseau qui permet aux containers d'être connecté à un réseau physique pour leur offrir des fonctionalités avancés de réseau en permettant une direct visibilité de couche Ethernet dans le modèle OSI. Son utilisation guarantit plus d'isolation en évitant les microVMs, qui représentent plus d'overhead. [24], [25]

	runc	Youki	crun	styrolite
Dépendance de la sécurité du kernel de l'hôte	Oui	Oui	Oui	Oui, mais dépassement de seccomp est impossible avec l'architecture du runtime ne dépendant pas sur seccomp
La mémoire utilisée par défaut par tmpfs est par défaut haute sauf si elle est configurée	Oui, le défaut est à 64MB, mais une image peut prendre plus d'espace sans limite [26]	Oui, le défaut est à 64MB, mais une image peut prendre plus d'espace sans limite [27]	Oui, le défaut est à 64MB, mais une image peut prendre plus d'espace sans limite [28]	Oui, car la fonctionalité n'est pas implémentée, le choix des paramètres de tmpfs est customisable
Incompatibilité avec Macvlan	Non [29]	Non, car il peut être rootful mais il n'est pas compatible avec Macvlan en mode rootless [30]	Oui, car il est rootless [31]	Non, il n'est pas rootless

Bibliography

[1] [Online]. Available: https://jameshunt.us/writings/dockerless-docker-with-runc/

[2] [Online]. Available: https://github.com/youki-dev/youki

[3] [Online]. Available: https://github.com/edera-dev/styrolite?tab=readme-ov-file

[4] [Online]. Available: https://www.lemondeinformatique.fr/actualites/lire-le-projet-open-source-styrolite-muscle-la-securite-du-runtime-des-conteneurs-96465.html

[5] [Online]. Available: https://www.youtube.com/watch?v=lHv0LVEIPk8&list=PLUVcZgyBBsZGrrtxGinRm7AaNHqJ_prCJ&index=3

[6] [Online]. Available: https://github.com/nabla-containers/runnc#limitations

[7] [Online]. Available: https://github.com/opencontainers/runc

[8] [Online]. Available: https://github.com/containers/crun/blob/main/src/libcrun/seccomp.c

[9] [Online]. Available: https://arxiv.org/pdf/2501.04580

[10] [Online]. Available: https://github.com/youki-dev/youki/blob/main/crates/libcontainer/src/apparmor.rs

[11] [Online]. Available: https://github.com/youki-dev/youki/tree/main/experiment/selinux/src

[12] [Online]. Available: https://www.pearsonitcertification.com/articles/article.aspx?p=2992608&seqNum=4

[13] [Online]. Available: https://www.infoq.com/news/2025/04/styrolite-low-level-runtime/?topicPageSponsorship=ce1390d7-0b6f-4b95-823c-b4b1f12acc26

[14] [Online]. Available: https://github.com/opencontainers/runc/pull/774

[15] [Online]. Available: https://man7.org/linux/man-pages/man2/seccomp.2.html

[16] [Online]. Available: https://doc.ubuntu-fr.org/apparmor

[17] [Online]. Available: https://docs.datadoghq.com/security/default_rules/byb-wyq-f3q/

[18] [Online]. Available: https://man7.org/linux/man-pages/man7/user_namespaces.7.html

[19] [Online]. Available: https://man7.org/linux/man-pages/man7/capabilities.7.html

[20] [Online]. Available: https://cyberveille.esante.gouv.fr/alertes/vulnerabilite-dans-le-noyau-de-linux-dirty-pipe-2022-09-29

[21] [Online]. Available: https://docs.docker.com/engine/storage/drivers/select-storage-drihttps://docs.docker.com/engine/storage/tmpfs/

[22] [Online]. Available: https://unix.stackexchange.com/questions/205595/what-happens-when-a-tmpfs-volume-is-full-and-swap-is-full-is-linuxs-oom-*killer

[23] [Online]. Available: https://docs.oracle.com/cd/E26502_01/html/E29014/conf-sec-fs-1.html

[24] [Online]. Available: https://docs.docker.com/engine/network/drivers/macvlan/#create-a-macvlan-network

[25] [Online]. Available: https://dev.to/rimelek/comparing-3-docker-container-runtimes-runc-gvisor-and-kata-containers-16j

[26] [Online]. Available: https://github.com/opencontainers/runc/blob/cdf953070162f400d75d2e0e83b506b3fb8cee5e/libcontainer/specconv/example.go#L68

[27] [Online]. Available: https://github.com/containers/crun/blob/9489b9962b2fa40e7c6b5d7260f40b3baa3cbd89/src/libcrun/linux.c#L1966

[28] [Online]. Available: https://github.com/youki-dev/youki/blob/61c9ae0c0ad13ae0bc165d9869633e2a82e08cac/crates/libcontainer/src/rootfs/utils.rs#L282

[29] [Online]. Available: https://github.com/opencontainers/runc/discussions/3411

[30] [Online]. Available: https://github.com/search?q=org%3Ayouki-dev+rootful&type=code

[31] [Online]. Available: https://github.com/containers/podman/issues/21867