Engineer

# Runtime de conteneurs

lancer et superviser un processus isolé depuis une image

## Ce que c’est

Un runtime de conteneurs est le logiciel qui se situe entre une image de conteneur sur disque et un processus isolé par le noyau. L’expression recouvre deux couches, et les personnes qui l’emploient ne parlent souvent pas de la même : le runtime de bas niveau, aussi appelé runtime OCI, et celui de haut niveau, plus souvent appelé moteur de conteneurs.

Le runtime de bas niveau accomplit une tâche bien délimitée. Il lit un bundle, que la spécification d’exécution de l’Open Container Initiative définit comme un répertoire contenant un fichier `config.json` et le système de fichiers racine du conteneur, demande au noyau les espaces de noms, cgroups et montages décrits dans la configuration, puis y démarre le processus. La spécification nomme les opérations que chaque runtime doit prendre en charge, à savoir state, create, start, kill et delete, et laisse la ligne de commande à chaque implémentation. La version 1.0 a été publiée en juillet 2017. runc, maintenu par l’OCI elle-même, se décrit comme « un outil de bas niveau qui n’a pas été conçu pour l’utilisateur final ». crun effectue le même travail en C avec une empreinte mémoire plus faible ; c’est le runtime préconfiguré indiqué par la documentation de Podman. youki en est un troisième, écrit en Rust.

Le moteur se situe au-dessus. containerd se décrit comme un démon qui gère tout le cycle de vie des conteneurs de son hôte : transfert et stockage des images, exécution et supervision des conteneurs, ainsi que leurs raccordements au stockage et au réseau. CRI-O accomplit un travail similaire, mais est conçu uniquement pour Kubernetes. Un moteur récupère et décompresse l’image, écrit le bundle, le confie au runtime de bas niveau, puis continue à surveiller le processus une fois ce runtime terminé. Docker Engine se situe encore un cran au-dessus : sa documentation indique qu’il utilise containerd pour le cycle de vie des conteneurs et que containerd utilise runc sans configuration contraire. Podman

suit une autre voie et ne conserve aucun démon permanent. Chaque commande prépare elle-même le bundle, appelle crun ou runc, puis confie la surveillance du conteneur en cours d'exécution à un petit moniteur nommé conmon.

Kubernetes ajoute un terme de plus. Sur chaque nœud, le kubelet communique avec le moteur par la Container Runtime Interface, ou CRI, un protocole gRPC implémenté par containerd et CRI-O. Docker Engine est antérieur à CRI ; Kubernetes a donc longtemps intégré pour lui un adaptateur nommé dockershim, supprimé dans la version v1.24.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce qu'un runtime de conteneurs ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Un logiciel conditionné pour se comporter à l'identique sur un portable, un serveur de test et en production, et monter en charge lors des semaines chargées.

**Découvrir nos prestations** Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. 20 applications Docker sur Google Cloud et AWS — 20 applications conteneurisées déployées et surveillées

**20 applications Docker sur Google Cloud et AWS** Contactez-nous →

## Pourquoi c'est important

Choisir un runtime de conteneurs revient à choisir la couche que l'on change, et les deux couches n'apportent pas les mêmes choses.

Le moteur décide du fonctionnement quotidien des conteneurs : qui récupère et stocke les images, qui redémarre un conteneur ayant planté, où vont ses journaux, quelle API les autres outils utilisent et qui gère l'arborescence des processus. Remplacer Docker Engine par Podman sur un serveur, ou par containerd sur un nœud Kubernetes, change tout cela sans changer les conteneurs eux-mêmes, car le format d'image et le bundle reposent sur les mêmes normes OCI. La documentation Kubernetes sur la suppression de dockershim le dit clairement : les images produites par `docker build` fonctionnent avec toutes les implémentations de CRI.

Le runtime de bas niveau décide du degré d'isolation du processus. runc et crun démarrent un processus ordinaire sur le noyau de l'hôte, ce qui est rapide et entièrement compatible, mais signifie que tous les conteneurs de la machine partagent ce même noyau. Le runtime de gVisor, runsc, place entre le conteneur et l'hôte un noyau applicatif écrit en Go et répond lui-même aux appels système

du conteneur ; sa documentation en précise le prix : une compatibilité applicative réduite et un surcoût par appel système. Kata Containers démarre chaque charge de travail dans une machine virtuelle légère dotée de son propre noyau, utilise la virtualisation matérielle comme seconde couche de défense et exige un hyperviseur sur chaque nœud qui l'exécute.

Kubernetes permet de faire ce second choix par pod plutôt que pour tout le cluster. Une RuntimeClass désigne un gestionnaire configuré dans le moteur, et un pod qui définit `runtimeClassName` s'exécute avec ce gestionnaire : une charge de travail à laquelle on ne fait pas confiance peut ainsi utiliser gVisor ou Kata tandis que tout le reste demeure sur runc. La documentation présente cela comme un équilibre entre performances et sécurité.

## À quoi cela ressemble en pratique

Dans l'usage courant, l'expression tend à désigner la couche du moteur, et les travaux présentés sur ce site l'emploient dans ce sens.

Vingt applications empaquetées avec Docker, réparties entre deux clouds et quelques runtimes de conteneurs, devaient se déployer de manière cohérente et permettre un diagnostic rapide en cas de panne. Docker servait à les empaqueter, Podman au dépannage et Kubernetes au débogage de l'orchestration. Travailler ainsi avec plusieurs moteurs s'appuie sur les couches décrites plus haut : les moteurs diffèrent, mais les normes OCI d'image et d'exécution qui les sous-tendent sont communes.

Sur un hôte plus récent, le choix s'est fait au niveau du moteur. La couche conteneurs fonctionne avec Podman et Quadlet à la place de Docker, parce que le moteur habituel publie les ports des conteneurs avec priorité sur les règles de pare-feu installées par le rôle de durcissement et ajouterait un second superviseur à côté de systemd. Le choix a changé qui démarre chaque conteneur et qui le surveille ensuite : les conteneurs sont devenus des services, démarrés comme les autres services, sous le superviseur auquel l'hôte faisait déjà confiance.

Le même raisonnement s'applique à un hôte de 512 MB pour les services de l'entreprise accessibles au public, où la mémoire a tranché tous les débats. Sa couche conteneurs consiste en des unités Quadlet sous systemd plutôt qu'en un démon doté de son propre superviseur, un choix fait sur une machine disposant d'environ 464 MB de mémoire utilisable.

# Où les problèmes surviennent

Les problèmes de runtime se manifestent souvent aux jonctions entre les couches, là où chacune suppose que l'autre est configurée comme elle l'attend.

**Les outils du nœud ne voient plus les conteneurs après une mise à niveau du cluster.** Depuis Kubernetes v1.24, le kubelet communique directement avec containerd ou CRI-O ; les conteneurs planifiés par le cluster ne passent donc jamais par Docker Engine. Sur le nœud, `docker ps` ne liste que ce que Docker a démarré lui-même, et les agents de supervision, scripts de démarrage et pods privilégiés qui appellent la CLI Docker ou montent `docker.sock` perdent leur visibilité. Les charges de travail continuent à s'exécuter. Ce sont les outils qui les entourent qu'il faut migrer vers `crictl` ou l'API du moteur.

**Le kubelet et le runtime gèrent les cgroups de deux façons différentes.** Les deux doivent utiliser le même pilote cgroup, `cgroupfs` ou `systemd`, et le kubelet utilise `cgroupfs` en l'absence d'autre configuration. Sur un hôte dont systemd est le système d'initialisation, cela laisse deux gestionnaires de cgroups avec deux visions des mêmes ressources ; la documentation Kubernetes avertit que ces nœuds peuvent devenir instables lorsque les ressources sont sous pression. La solution consiste à utiliser le pilote systemd des deux côtés, ce qui est aussi recommandé avec cgroup v2. Avec containerd 2.x et runc, la configuration du runtime ressemble à ceci :

```
# /etc/containerd/config.toml, containerd 2.x
[plugins.'io.containerd.cri.v1.runtime'.containerd.runtimes.runc.options]
  SystemdCgroup = true
```

Côté kubelet, il faut `cgroupDriver: systemd` dans sa configuration, et containerd doit ensuite être redémarré. containerd 1.x écrit la section différemment ; la page Kubernetes donne les deux formes. Changer de pilote sur un nœud qui a déjà rejoint un cluster est délicat : des pods créés sous un pilote peuvent rencontrer des erreurs lorsqu'ils sont recréés sous l'autre. La documentation conseille donc de remplacer ou de réinstaller le nœud avec la nouvelle configuration.

**Un pod déplacé vers un runtime en bac à sable s'arrête au démarrage.** Une RuntimeClass dont le gestionnaire manque dans le moteur du nœud envoie directement le pod dans la phase Failed, avec la raison dans un événement. Ce

point réglé, la charge doit encore convenir au bac à sable. gVisor répond aux appels système dans son propre noyau ; les logiciels qui s'appuient sur des fonctions du noyau moins courantes peuvent donc se comporter différemment, et sa documentation juge les charges qui effectuent de nombreux appels système mal adaptées. Kata exige la virtualisation matérielle sur le nœud, ce qui, sur une machine virtuelle cloud, signifie une virtualisation imbriquée. Un sélecteur de nœuds dans `scheduling` de la RuntimeClass maintient ces pods sur les nœuds capables de les exécuter.

**Une image importée à la main reste invisible pour le kubelet.** containerd conserve les noms d'images et leurs métadonnées par espace de noms. Son client `ctr` utilise l'espace de noms `default` sauf indication contraire, tandis que le plugin CRI auquel parle le kubelet utilise `k8s.io` . Une image chargée avec un simple `ctr images import` arrive donc là où le kubelet ne regarde jamais ; le pod retourne au registre ou s'arrête sur une erreur de récupération d'image. Importer avec `ctr -n k8s.io images import` et vérifier avec `crictl images` , qui voit ce que voit le kubelet, évite cet écart.

## Pour aller plus loin

- Spécification d'exécution OCI — la norme implémentée par chaque runtime de bas niveau : le bundle, `config.json` et les opérations du cycle de vie.
- runc — le runtime de l'OCI elle-même, avec des notes sur la manière dont les outils de plus haut niveau le pilotent.
- Runtimes de conteneurs — pilotes cgroup, versions CRI et réglages de containerd et CRI-O attendus par Kubernetes.
- Container Runtime Interface (CRI) — le protocole gRPC entre le kubelet et le moteur.
- Runtime Class — choisir un gestionnaire d'exécution différent par pod et maintenir ces pods sur les nœuds qui le prennent en charge.
- Runtimes de conteneurs alternatifs — comment Docker Engine confie des conteneurs à gVisor, Kata Containers, youki ou Wasmtime.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le parcours de déploiement, les risques opérationnels et les personnes d'astreinte. Nous pouvons définir un changement avec des vérifications, une procédure de reprise et une responsabilité claire.

**Consulter les tarifs** Contactez-nous →

# Rendez la terminologie concrète

Demandez comment « Qu'est-ce qu'un runtime de conteneurs ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

**Découvrir nos prestations**

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/container-runtime/