Engineer

Container-runtime

programmet, der faktisk starter en container

Hvad det er

En container-runtime er softwaren mellem et containerimage på disken og en isoleret proces i kernen. Ordet dækker to lag, og folk mener ofte hvert sit: den lave runtime, også kaldet OCI-runtimen, og den høje runtime, som oftere hedder en containermotor. I flertal hedder det runtimes.

Den lave runtime har én snæver opgave. Den læser et bundle, som Open Container Initiatives runtime-specifikation definerer som en mappe med en `config.json` og containerens rodfilsystem, beder kernen om de navnerum, cgroups og mounts, konfigurationen beskriver, og starter processen inden i dem. Specifikationen fastlægger de operationer, enhver runtime skal understøtte, nemlig create, start, kill, delete og en forespørgsel på tilstand, og overlader kommandolinjen til den enkelte implementering. Version 1.0 udkom i juli 2017. runc, som OCI selv vedligeholder, beskriver sig som et værktøj på lavt niveau, der ikke er tænkt til slutbrugere. crun gør det samme arbejde, er skrevet i C med et mindre hukommelsesforbrug og er den standard, Podmans dokumentation nævner. youki er en tredje, skrevet i Rust.

Motoren ligger ovenpå. containerd beskriver sig som en dæmon, der styrer hele containerens livscyklus på sin vært: overførsel og lagring af images, kørsel og overvågning af containere og den lagring og netværkstilkobling, der hører til. CRI-O gør et lignende arbejde, bygget alene til Kubernetes. En motor henter og pakker imaget ud, skriver bundlet, giver det videre til den lave runtime og bliver ved med at holde øje med processen, også efter at runtimen selv er afsluttet. Docker Engine ligger et trin længere oppe: ifølge dokumentationen bruger den containerd til containerens livscyklus, og containerd bruger som standard runc. Podman går den anden vej og holder ingen dæmon kørende. Hver kommando forbereder selv bundlet, kalder crun eller runc og overlader den kørende container til en lille overvågningsproces ved navn conmon.

Kubernetes lægger et ord mere til. Kubelet på hver node taler med motoren over Container Runtime Interface, CRI, en gRPC-protokol, som både containerd og CRI-O implementerer. Docker Engine er ældre end CRI, så Kubernetes havde i årevis en adapter til den ved navn dockershim og fjernede den i v1.24.

Brug begrebet i en beslutning

Brug »Hvad er en container-runtime?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Pakket, så den opfører sig identisk på en bærbar, en testserver og i produktion, og skalerer op, når der kommer en travl uge.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har drevet 20 Docker-apps på Google Cloud og AWS — 20 containeriserede applikationer udrullet og overvåget

Har drevet 20 Docker-apps på Google Cloud og AWS Kontakt os →

Hvorfor det betyder noget

At vælge container-runtime er i virkeligheden at vælge, hvilket lag man ændrer, og de to lag giver forskellige ting.

Motoren afgør, hvordan containere lever fra dag til dag: hvem der henter og gemmer images, hvem der genstarter en container, der er gået ned, hvor dens logs ender, hvilket API andre værktøjer taler, og hvem der ejer procestræet. Skifter man Docker Engine ud med Podman på en server eller med containerd på en Kubernetes-node, ændrer det alt dette og lader selve containerne være, fordi imageformatet og bundlet bygger på de samme OCI-standarder nedenunder. Kubernetes-dokumentationen om fjernelsen af dockershim siger det ligeud: images bygget med `docker build` virker med alle CRI-implementeringer.

Den lave runtime afgør, hvor isoleret processen er. runc og crun starter en almindelig proces på værtens kerne. Det er hurtigt og fuldt kompatibelt, og det betyder, at alle containere på maskinen deler den samme kerne. gVisors runtime, runsc, sætter en applikationskerne skrevet i Go ind mellem containeren og værten og besvarer selv containerens systemkald; projektets egen dokumentation nævner prisen som ringere kompatibilitet og større omkostning pr. systemkald. Kata Containers starter hver workload i en letvægts virtuel maskine med sin egen kerne, bruger hardwarevirtualisering som et ekstra forsvarslag og kræver en hypervisor på hver node, der kører den.

Kubernetes lader det andet valg træffes pr. pod i stedet for pr. klynge. En RuntimeClass peger på en handler, der er sat op i motoren, og en pod med `runtimeClassName` kører under den handler. Så kan en workload, man ikke stoler på, ligge i gVisor eller Kata, mens resten bliver på runc. Dokumentationen beskriver det som en afvejning mellem ydeevne og sikkerhed.

Sådan ser det ud i praksis

I daglig tale glider ordet over mod motorlaget, og arbejdet på dette site bruger det i den betydning.

Tyve Docker-pakkede applikationer, spredt over to clouds og et par container-runtimes, skulle udrulles ensartet, og fejl i dem skulle kunne diagnosticeres hurtigt. Docker pakkede dem, fejlfindingen gik gennem Podman, og orkestreringsdebugging gennem Kubernetes. At arbejde på tværs af motorer på den måde hviler på lagdelingen ovenfor: motorerne er forskellige, og OCI-standarderne for images og runtimes under dem er fælles.

På en nyere vært blev valget truffet i motorlaget. Containerplanet kører på Podman og Quadlet i stedet for Docker, fordi standardmotoren publicerer containerporte over de firewallregler, hærdningsrollen installerer, og fordi den ville lægge en ekstra supervisor ved siden af systemd. Det, valget ændrede, er, hvem der starter hver container, og hvem der holder øje med den bagefter: containerne blev tjenester, der starter, som tjenester starter, under den supervisor, værten i forvejen stolede på.

Samme ræsonnement gælder en vært med 512 MB til virksomhedens offentligt tilgængelige tjenester, hvor hukommelsen afgjorde hver diskussion. Dens containerlag er Quadlet-units under systemd frem for en dæmon med sin egen supervisor, et valg truffet på en maskine, hvor omkring 464 MB hukommelse kan bruges.

Hvor det går galt

Problemer med runtimes dukker typisk op i sømmene mellem lagene, hvor hver side går ud fra, at den anden er sat op, som den forventer.

Værktøjerne på noden går i blinde efter en klyngeopgradering. Fra Kubernetes v1.24 taler kubelet direkte med containerd eller CRI-O, så de containere, klyngen planlægger, går uden om Docker Engine. `docker ps` på noden viser kun det, Docker selv har startet, og overvågningsagenter, opstartsscripts og privilegerede pods, der kalder Docker-CLI'en eller monterer `docker.sock`, mister overblikket. Workloads kører videre. Det er værktøjerne omkring dem, der skal flyttes over på `crictl` eller motorens eget API.

Kubelet og runtimen styrer cgroups på hver sin måde. Begge skal bruge den samme cgroup-driver, `cgroupfs` eller `systemd`, og kubelet bruger som standard `cgroupfs`. På en vært, hvor systemd er init-systemet, giver det to cgroup-styringer med hvert sit billede af de samme ressourcer, og Kubernetes-dokumentationen advarer om, at sådanne noder kan blive ustabile under ressourcepres. Løsningen er systemd-driveren på begge sider, som også er den anbefalede driver med cgroup v2. For containerd 2.x med runc ser runtimesiden sådan ud:

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

På kubelets side er det `cgroupDriver: systemd` i konfigurationen, og containerd skal genstartes bagefter. containerd 1.x staver sektionen anderledes, og Kubernetes-siden viser begge former. At skifte driver på en node, der allerede er med i en klynge, er følsomt: pods, der er oprettet under den ene driver, kan løbe ind i fejl, når de oprettes igen under den anden, så det dokumenterede råd er at udskifte eller geninstallere noden med den nye konfiguration.

En pod, der flyttes til en sandboxet runtime, stopper ved opstart. En RuntimeClass, hvis handler nodens motor mangler, sender podden direkte i fasen Failed med årsagen i en event. Derudover skal workloaden passe til sandkassen. gVisor besvarer systemkald i sin egen kerne, så software, der læner sig op ad mindre almindelige kernefunktioner, kan opføre sig anderledes, og dokumentationen nævner workloads med mange systemkald som et dårligt match. Kata kræver hardwarevirtualisering på noden, hvilket på en virtuel maskine i skyen betyder indlejret virtualisering. En `scheduling`-nodeselector på RuntimeClass'en holder sådanne pods på de noder, der kan køre dem.

Et image, der er importeret i hånden, forbliver usynligt for kubelet. containerd holder imagenavne og metadata pr. navnerum. Klienten `ctr` arbejder i navnerummet `default`, medmindre man angiver et andet, mens CRI-pluginet, som kubelet taler med, arbejder i `k8s.io`. Et image indlæst med en almindelig `ctr images import` lander derfor et sted, kubelet aldrig kigger, og podden går tilbage til registret eller stopper med en fejl ved image-pull. Import med `ctr -n k8s.io images import` og kontrol med `crictl images`, der ser det samme som kubelet, lukker hullet.

Læs mere

- OCI Runtime Specification — standarden, som enhver lav runtime implementerer: bundlet, `config.json` og livscyklusoperationerne.
- runc — OCI's egen runtime med noter om, hvordan værktøjer på højere niveau styrer den.
- Container Runtimes — cgroup-drivere, CRI-versioner og de indstillinger for containerd og CRI-O, Kubernetes forventer.
- Container Runtime Interface (CRI) — gRPC-protokollen mellem kubelet og motoren.
- Runtime Class — valg af en anden runtime-handler pr. pod, og hvordan de pods holdes på noder, der understøtter den.
- Alternative container runtimes — hvordan Docker Engine giver containere videre til gVisor, Kata Containers, youki eller Wasmtime.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring udrulningsforløbet, driftsrisiciene og dem, der har vagten. Vi kan afgrænse en ændring med kontroller, en vej tilbage og tydeligt ansvar.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er en container-runtime?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

welcome@engineer.company +45 50 10 61 84 København, Danmark

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/container-runtime/