Engineer

Containerkjøretid

programmet som faktisk starter en container

Hva det er

En containerkjøretid er programvaren mellom et containerimage på disk og en isolert prosess i systemkjernen. Ordet dekker to lag, og folk mener ofte hvert sitt: kjøretiden på lavt nivå, også kalt OCI-kjøretiden, og kjøretiden på høyt nivå, som oftere kalles en containermotor.

Kjøretiden på lavt nivå gjør én smal jobb. Den leser en bundle, som kjøretidsspesifikasjonen fra Open Container Initiative definerer som en katalog med en `config.json`-fil og containerens rotfilsystem, ber kjernen om de navnerommene (namespaces), cgroups og monteringene som konfigurasjonen beskriver, og starter prosessen inni dem. Spesifikasjonen navngir operasjonene hver kjøretid må støtte, nemlig create, start, kill, delete og en forespørsel om tilstand, og overlater kommandolinjen til hver implementasjon. Versjon 1.0 ble publisert i juli 2017. runc, som OCI selv vedlikeholder, beskriver seg som et verktøy på lavt nivå som ikke er laget med tanke på en sluttbruker. crun gjør den samme jobben i C med mindre minnebruk og er standardvalget som dokumentasjonen til Podman navngir. youki er en tredje, skrevet i Rust.

Motoren ligger over den. containerd beskriver seg som en daemon som styrer hele livssyklusen til containerne på verten sin: overføring og lagring av images, kjøring av og tilsyn med containere, og lagrings- og nettverkstilkoblingene rundt dem. CRI-O gjør en lignende jobb, bygd for Kubernetes alene. En motor henter og pakker ut imaget, skriver bundlen, gir den videre til kjøretiden på lavt nivå og fortsetter å følge med på prosessen etter at den kjøretiden har avsluttet. Docker Engine ligger enda ett trinn høyere: dokumentasjonen sier at den bruker containerd til livssyklusen for containere, og at containerd bruker runc som standard. Podman går den andre veien og holder ingen daemon kjørende hele tiden. Hver kommando forbereder bundlen selv, kaller crun eller runc og overlater den kjørende containeren til en liten monitor som heter conmon.

Kubernetes føyer ett begrep til vokabularet. Kubelet på hver node snakker med motoren over Container Runtime Interface, CRI, en gRPC-protokoll som både containerd og CRI-O implementerer. Docker Engine er eldre enn CRI, så i årevis hadde Kubernetes med seg en adapter for den, kalt dockershim, og fjernet den i v1.24.

Bruk begrepet i en beslutning

Bruk «Hva er en containerkjøretid?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Pakket slik at den oppfører seg helt likt på en bærbar PC, på en testserver og i produksjon, og skalerer opp når en travel uke kommer.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Driftet 20 Docker-apper på Google Cloud og AWS — 20 containeriserte applikasjoner produksjonssatt og overvåket

Driftet 20 Docker-apper på Google Cloud og AWS Ta kontakt →

Hvorfor det er viktig

Å velge en containerkjøretid er egentlig å velge hvilket lag du endrer, og de to lagene gir deg ulike ting.

Motoren avgjør hvordan containere lever fra dag til dag: hvem som henter og lagrer images, hvem som starter en krasjet container på nytt, hvor loggene havner, hvilket API andre verktøy snakker, og hvem som eier prosesstreet. Å bytte Docker Engine mot Podman på en server, eller mot containerd på en Kubernetes-node, endrer alt dette og lar selve containerne være i fred, fordi imageformatet og bundlen bygger på de samme OCI-standardene under. Kubernetes-dokumentasjonen om fjerningen av dockershim sier det rett ut: images bygd med `docker build` fungerer med alle CRI-implementasjoner.

Kjøretiden på lavt nivå avgjør hvor isolert prosessen er. runc og crun starter en vanlig prosess på vertens kjerne, noe som er raskt og fullt kompatibelt, og som betyr at alle containerne på maskinen deler den ene kjernen. Kjøretiden til gVisor, runsc, setter en applikasjonskjerne skrevet i Go mellom containeren og verten og besvarer containerens systemkall selv. Dokumentasjonen til gVisor oppgir prisen som redusert applikasjonskompatibilitet og høyere kostnad per systemkall. Kata Containers starter hver arbeidslast i en lett virtuell maskin med en egen kjerne, bruker maskinvarevirtualisering som et ekstra forsvarslag og trenger en hypervisor på hver node som kjører den.

Kubernetes lar det andre valget tas per pod i stedet for per klynge. En RuntimeClass navngir en handler som er satt opp i motoren, og en pod som setter `runtimeClassName`, kjører under den handleren. Slik kan en arbeidslast du ikke stoler på, ligge i gVisor eller Kata mens alt annet blir på runc. Dokumentasjonen beskriver dette som en avveining mellom ytelse og sikkerhet.

Slik ser det ut i praksis

I dagligtale glir uttrykket mot motorlaget, og arbeidet på dette nettstedet bruker det i den betydningen.

Tjue applikasjoner pakket med Docker, spredt over to skyer og noen få containerkjøretider, måtte produksjonssettes konsekvent og kunne diagnostiseres raskt når noe gikk galt. Docker pakket dem, feilsøkingen gikk gjennom Podman, og feilsøkingen av orkestreringen gikk gjennom Kubernetes. Å arbeide på tvers av motorer på den måten støtter seg på lagdelingen ovenfor: motorene er ulike, og OCI-standardene for image og kjøretid under dem er felles.

På en nyere vert ble valget tatt i motorlaget. Containerlaget kjører på Podman og Quadlet i stedet for Docker, fordi standardmotoren publiserer containerporter over brannmurreglene som herdingsrollen installerer, og fordi den ville lagt til en ekstra tjenestestyrer ved siden av systemd. Det valget endret, er hvem som starter hver container, og hvem som følger med på den etterpå: containere ble tjenester som starter slik tjenester starter, under den tjenestestyreren verten allerede stolte på.

Den samme tankegangen gjelder for en vert med 512 MB til selskapets offentlig tilgjengelige tjenester, der minnet avgjorde hver diskusjon. Containerlaget er Quadlet-enheter under systemd i stedet for en daemon med sin egen tjenestestyrer, et valg tatt på en maskin der omtrent 464 MB minne er brukbart.

Hvor det går galt

Problemer med kjøretiden dukker gjerne opp i skjøtene mellom lagene, der hver side antar at den andre er satt opp slik den forventer.

Verktøyene på noden mister oversikten etter en klyngeoppgradering. Fra Kubernetes v1.24 snakker kubelet direkte med containerd eller CRI-O, så containerne klyngen planlegger, går aldri gjennom Docker Engine. `docker ps` på

noden lister bare det Docker selv har startet, og overvåkingsagenter, oppstartsskript og privilegerte pods som kaller kommandolinjeverktøyet til Docker eller monterer `docker.sock`, mister innsynet. Arbeidslastene fortsetter å kjøre. Det er verktøyene rundt dem som må flyttes over til `crictl` eller motorens eget API.

Kubelet og kjøretiden styrer cgroups på hver sin måte. Begge må bruke den samme cgroup-driveren, `cgroupfs` eller `systemd`, og kubelet bruker `cgroupfs` som standard. På en vert der systemd er init-systemet, gir det to cgroup-styrere med to bilder av de samme ressursene, og Kubernetes-dokumentasjonen advarer om at slike noder kan bli ustabile under ressurspress. Rettelsen er systemd-driveren på begge sider, som også er den anbefalte driveren med cgroup v2. For containerd 2.x med runc ser kjøretidssiden slik ut:

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

Kubelet-siden er `cgroupDriver: systemd` i konfigurasjonen, og containerd må startes på nytt etterpå. containerd 1.x staver seksjonen annerledes, og Kubernetes-siden gir begge formene. Å bytte driver på en node som allerede er med i en klynge, er ømfintlig: pods opprettet under én driver kan få feil når de opprettes på nytt under den andre, så det dokumenterte rådet er å erstatte noden eller installere den på nytt med den nye konfigurasjonen.

En pod som flyttes til en sandkassekjøretid, stopper ved oppstart. En RuntimeClass med en handler som nodens motor ikke har, sender poden rett til fasen Failed, med årsaken i en hendelse. Selv med handleren på plass må arbeidslasten passe til sandkassen. gVisor besvarer systemkall i sin egen kjerne, så programvare som støtter seg på mindre vanlige kjernefunksjoner, kan oppføre seg annerledes, og dokumentasjonen nevner arbeidslaster med mange systemkall som en dårlig match. Kata trenger maskinvarevirtualisering på noden, og på en virtuell maskin i skyen betyr det nestet virtualisering. En nodevelger i feltet `scheduling` på RuntimeClass-en holder slike pods på nodene som kan kjøre dem.

Et image importert for hånd forblir usynlig for kubelet. containerd holder navn og metadata for images per navnerom. Klienten `ctr` arbeider i navnerommet `default` hvis ikke noe annet er oppgitt, mens CRI-pluginen som kubelet snakker

med, arbeider i `k8s.io`. Et image som lastes inn med en vanlig `ctr images import`, havner derfor der kubelet aldri ser, og poden går tilbake til registeret eller stopper med en feil ved henting av imaget. Å importere med `ctr -n k8s.io images import`, og å sjekke med `crictl images`, som ser det kubelet ser, lukker gapet.

Les mer

- OCI Runtime Specification — standarden hver kjøretid på lavt nivå implementerer: bundlen, `config.json` og operasjonene i livssyklusen.
- runc — OCIs egen kjøretid, med notater om hvordan verktøy på høyere nivå styrer den.
- Container Runtimes — cgroup-drivere, CRI-versjoner og innstillingene for containerd og CRI-O som Kubernetes forventer.
- Container Runtime Interface (CRI) — gRPC-protokollen mellom kubelet og motoren.
- Runtime Class — valg av en annen handler per pod, og hvordan slike pods holdes på noder som støtter den.
- Alternative container runtimes — hvordan Docker Engine gir containere videre til gVisor, Kata Containers, youki eller Wasmtime.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er en containerkjøretid?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/container-runtime/