Engineer

Containerkörmiljö

programmet som faktiskt startar en container

Vad det är

En containerkörmiljö är programvaran som står mellan en containeravbildning på disken och en isolerad process i kärnan. På engelska heter den container runtime, och i svenskt tal hörs ofta bara runtime eller körmiljö. Uttrycket täcker två lager, och folk menar ofta olika saker med det: körmiljön på låg nivå, som också kallas OCI-körmiljön, och körmiljön på hög nivå, som oftare kallas containermotor (container engine).

Körmiljön på låg nivå gör ett enda smalt jobb. Den läser in ett paket, i specifikationen kallat bundle, som körmiljöspecifikationen från Open Container Initiative definierar som en katalog med en `config.json`-fil och containerns rotfilsystem. Den ber kärnan om de namnrymder, cgroups och monteringar som konfigurationen beskriver och startar processen inuti dem. Specifikationen namnger de operationer som varje körmiljö måste stödja, nämligen create, start, kill, delete och en fråga om tillståndet, och lämnar kommandoraden åt varje implementation. Version 1.0 publicerades i juli 2017. runc, som OCI självt underhåller, beskriver sig som ett verktyg på låg nivå som inte är tänkt för en slutanvändare. crun gör samma jobb i C med ett mindre minnesavtryck och är den standard som Podmans dokumentation anger. youki är en tredje, skriven i Rust.

Motorn ligger ovanpå. containerd beskriver sig som en daemon som hanterar hela livscykeln för containrarna på sin maskin: överföring och lagring av avbildningar, körning och tillsyn av containrar och de lagrings- och nätverkskopplingar som hör till dem. CRI-O gör ett liknande jobb, byggt enbart för Kubernetes. En motor hämtar och packar upp avbildningen, skriver paketet, lämnar över det till körmiljön på låg nivå och fortsätter att bevaka processen efter att den körmiljön har avslutats. Docker Engine ligger ytterligare ett steg högre: enligt dess dokumentation använder den containerd för containrarnas livscykel, och containerd använder i sin tur runc som standard. Podman går den

andra vägen och har ingen daemon som körs hela tiden. Varje kommando förbereder paketet självt, anropar crun eller runc och lämnar en liten övervakare som heter conmon att ta hand om den körande containern.

Kubernetes lägger till ytterligare ett ord i vokabulären. Kubelet på varje nod talar med motorn via Container Runtime Interface, CRI, ett gRPC-protokoll som både containerd och CRI-O implementerar. Docker Engine är äldre än CRI, så i flera år bar Kubernetes med sig en adapter för den som hette dockershim, och den togs bort i v1.24.

Använd begreppet i ett beslut

Använd ”Vad är en containerkörmiljö?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Paketerad för att bete sig identiskt på en laptop, på en testserver och i produktion, och för att skala upp när en hektisk vecka kommer.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Drev 20 Docker-appar på Google Cloud och AWS — 20 containerbaserade applikationer, driftsatta och övervakade

Drev 20 Docker-appar på Google Cloud och AWS Hör av er →

Varför det spelar roll

Att välja containerkörmiljö är i själva verket att välja vilket lager man byter, och de två lagren ger olika saker.

Motorn avgör hur containrarna lever från dag till dag: vem som hämtar och lagrar avbildningar, vem som startar om en container som har kraschat, vart dess loggar tar vägen, vilket API andra verktyg talar och vem som äger processträdet. Att byta Docker Engine mot Podman på en server, eller mot containerd på en nod i Kubernetes, ändrar allt detta och lämnar själva containrarna orörda, eftersom avbildningsformatet och paketet bygger på samma OCI-standarder därunder. Kubernetes dokumentation om borttagningen av dockershim säger det rakt ut: avbildningar som har byggts med `docker build` fungerar med alla CRI-implementationer.

Körmiljön på låg nivå avgör hur isolerad processen är. runc och crun startar en vanlig process på maskinens egen kärna, vilket är snabbt och helt kompatibelt och betyder att varje container på maskinen delar just den kärnan. gVisors körmiljö, runsc, lägger en applikationskärna skriven i Go mellan containern och maskinen och besvarar containerns systemanrop själv. Projektets egen dokumentation anger priset som sämre kompatibilitet för applikationer och

högre overhead per systemanrop. Kata Containers startar varje arbetslast i en lätt virtuell maskin med en egen kärna, använder hårdvaruvirtualisering som ett andra försvarslager och behöver en hypervisor på varje nod som kör den.

Kubernetes låter det andra valet göras per pod i stället för per kluster. En RuntimeClass namnger en hanterare som är konfigurerad i motorn, och en pod som sätter `runtimeClassName` körs under den hanteraren, så att en arbetslast man inte litar på kan ligga i gVisor eller Kata medan allt annat stannar på runc. Dokumentationen beskriver detta som en avvägning mellan prestanda och säkerhet.

Så ser det ut i praktiken

I vardagligt tal glider uttrycket mot motorlagret, och arbetet på den här webbplatsen använder det i den betydelsen.

Tjugo applikationer paketerade med Docker, utspridda över två moln och några körmiljöer för containrar, behövde driftsättas konsekvent och kunna felsökas snabbt när något gick sönder. Docker paketerade dem, felsökningen gick genom Podman och felsökningen av orkestreringen genom Kubernetes. Att arbeta över flera motorer på det sättet vilar på lagerindelningen ovan: motorerna skiljer sig åt, men OCI-standarderna för avbildningar och körmiljöer under dem är gemensamma.

På en nyare server gjordes valet i motorlagret. Containerskiktet körs på Podman och Quadlet i stället för Docker, eftersom standardmotorn publicerar containerportar ovanför de brandväggsregler som härdningsrollen installerar, och eftersom den skulle lägga till en andra tillsynsprocess bredvid systemd. Det valet ändrade är vem som startar varje container och vem som bevakar den efteråt: containrarna blev tjänster som startar på samma sätt som tjänster startar, under den tillsyn som maskinen redan litade på.

Samma resonemang gäller för en värddator med 512 MB för företagets publika tjänster, där minnet avgjorde varje diskussion. Containerskiktet där är Quadlet-enheter under systemd snarare än en daemon med en egen tillsynsprocess, ett val som gjordes på en maskin där ungefär 464 MB minne går att använda.

Var det går fel

Problem med körmiljön brukar dyka upp i skarvarna mellan lagren, där varje sida utgår från att den andra är uppsatt på det sätt den själv väntar sig.

Verktygen på noden tappar sikten efter en uppgradering av klustret. Från och med Kubernetes v1.24 talar kubelet direkt med containerd eller CRI-O, så de containrar som klustret startar passerar aldrig genom Docker Engine. `docker ps` på noden listar bara det som Docker själv har startat, och övervakningsagenter, startskript och privilegierade poddar som anropar Dockers CLI eller monterar `docker.sock` förlorar sin överblick. Arbetslasterna fortsätter att köras. Det är verktygen runt dem som behöver flyttas till `crictl` eller till motorns eget API.

Kubelet och körmiljön hanterar cgroups på två olika sätt. Båda måste använda samma cgroup-drivrutin, `cgroupfs` eller `systemd`, och kubelet använder `cgroupfs` som standard. På en maskin där systemd är init-systemet blir det två hanterare av cgroups med två bilder av samma resurser, och Kubernetes dokumentation varnar för att sådana noder kan bli instabila när resurserna tryter. Lösningen är systemd-drivrutinen på båda sidor, som också är den rekommenderade drivrutinen med cgroup v2. För containerd 2.x med runc ser körmiljöns sida ut så här:

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

Kubelets sida är `cgroupDriver: systemd` i dess konfiguration, och containerd behöver startas om efteråt. containerd 1.x stavar avsnittet annorlunda, och Kubernetes sida visar båda formerna. Att byta drivrutin på en nod som redan har gått med i ett kluster är känsligt: poddar som skapades under den ena drivrutinen kan stöta på fel när de skapas på nytt under den andra, så det dokumenterade rådet är att ersätta eller installera om noden med den nya konfigurationen.

En pod som flyttas till en sandlådad körmiljö stannar vid start. En RuntimeClass vars hanterare nodens motor saknar skickar podden direkt till fasen Failed, med orsaken i en händelse. Därefter måste arbetslasten passa sandlådan. gVisor besvarar systemanrop i sin egen kärna, så programvara som lutar sig mot mindre vanliga funktioner i kärnan kan bete sig annorlunda, och

dokumentationen pekar ut arbetslaster med många systemanrop som en dålig matchning. Kata behöver hårdvaruvirtualisering på noden, vilket på en virtuell maskin i molnet betyder nästlad virtualisering. En nodväljare under `scheduling` i RuntimeClass håller sådana poddar på de noder som kan köra dem.

En avbildning som har importerats för hand syns aldrig för kubelet. containerd håller namn och metadata för avbildningar per namnrymd. Klienten `ctr` arbetar i namnrymden `default` om inget annat anges, medan CRI-insticksmodulen som kubelet talar med arbetar i `k8s.io`. En avbildning som läses in med ett vanligt `ctr images import` hamnar därför där kubelet aldrig tittar, och podden går tillbaka till registret eller stannar med ett fel vid hämtningen av avbildningen. Att importera med `ctr -n k8s.io images import` och kontrollera med `crictl images`, som ser det som kubelet ser, täpper till glappet.

Läs mer

- OCI Runtime Specification — standarden som varje körmiljö på låg nivå implementerar: paketet, `config.json` och operationerna i livscykeln.
- runc — OCI:s egen körmiljö, med anteckningar om hur verktyg på högre nivå styr den.
- Container Runtimes — cgroup-drivrutiner, CRI-versioner och de inställningar för containerd och CRI-O som Kubernetes förväntar sig.
- Container Runtime Interface (CRI) — gRPC-protokollet mellan kubelet och motorn.
- Runtime Class — att välja en annan hanterare för körmiljön per pod och hålla de poddarna på noder som stöder den.
- Alternative container runtimes — hur Docker Engine lämnar över containrar till gVisor, Kata Containers, youki eller Wasmtime.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är en containerkörmiljö?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/container-runtime/