Engineer

Docker

ett sätt att leverera ett program med allt det behöver

Vad det är

Docker är en uppsättning verktyg för att packa ett program, dess bibliotek, körmiljön för dess språk och dess filer i en enda artefakt som kallas avbildning, på engelska image, och sedan köra avbildningen som en isolerad process som kallas container. Avbildningen byggs en gång från ett recept, en Dockerfile, och samma avbildning startar sedan på en bärbar dator, på en byggserver och i produktion. Det är hela löftet i en mening: det som testades är byte för byte det som körs.

Fyra substantiv bär det mesta av ordförrådet. En Dockerfile är en textfil med instruktioner som `FROM`, `COPY` och `RUN`. En avbildning är en skrivskyddad mall som byggs från de instruktionerna och lagras som en stapel av lager, ett per instruktion, så att ett nytt bygge efter en liten ändring bara gör om de lager som ändrades. En container är en körande instans av en avbildning med ett tunt skrivbart lager överst. Ett register, som Docker Hub eller molnleverantörens eget, är platsen dit avbildningar pushas och varifrån de hämtas.

Under kommandoraden finns en klient och en server. Kommandot `docker` är klienten; det skickar anrop över ett REST-API, oftast genom en Unix-socket, till en långlivad daemon som heter `dockerd`. Daemonen äger avbildningar, containrar, nätverk och volymer. För själva livscykeln hos en container lämnar den arbetet vidare till containerd, och containerd använder som standard runc, det lilla programmet som ber Linux-kärnan om namnrymder och kontrollgrupper och startar processen inuti dem. En container är alltså en vanlig process på värddatorns systemkärna, med en begränsad vy av resten av systemet. Den är ingen virtuell maskin, och på macOS och Windows kör Docker Desktop en liten virtuell Linux-maskin just för att ha en Linux-kärna att dela.

Avbildningsformatet har vuxit ifrån företaget som skapade det. I juni 2015 startade Docker, CoreOS och andra Open Container Initiative, Docker skänkte sitt avbildningsformat och runc till initiativet, och de specifikationer som blev resultatet är skälet till att en avbildning som har byggts med Docker körs oförändrad under Podman, containerd eller en hanterad molntjänst. Docker är i dag ett av flera verktyg som talar samma format, och ordet används ofta löst om hela området.

Använd begreppet i ett beslut

Använd ”Vad är Docker?” 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

Den användbara frågan är aldrig ”Docker eller inte” i det abstrakta. Den är om en viss arbetslast är värd en avbildning, eller om en vanlig tjänst direkt på värddatorn skulle göra samma jobb med färre rörliga delar.

En avbildning gör skäl för sin plats när programmet drar en hel miljö efter sig. En applikation i Python eller R som behöver en exakt uppsättning paket, en äldre tjänst som är låst till en gammal körmiljö, två program som vill ha olika versioner av samma bibliotek: vart och ett är mödosamt att återskapa för hand och enkelt att återskapa från en avbildning. Att rulla tillbaka innebär att starta den föregående avbildningen. En ny utvecklare kör ett enda kommando i stället för att följa en sida med installationsanteckningar. Docker jämfört med en virtuell maskin handlar om samma avvägning sedd från andra hållet: containrar startar på ett ögonblick och delar en systemkärna, så dussintals ryms på en värddator som skulle rymma en handfull virtuella maskiner, till priset av en svagare isolering.

Kostnaden är verklig infrastruktur. Som standard körs daemonen som root och hanterar nätverket; Docker stöder också rootless-läge. Det finns ett register att driva eller betala för, basavbildningar som samlar på sig säkerhetsrättningar och

måste byggas om för att få dem, och ett lagringsskikt vars regler skiljer sig från värddatorns. Ett enda statiskt program utan beroenden vinner väldigt lite på allt det, och en systemd-enhet i en sandlåda kör det lika bra.

Skillnaden mellan Docker och Podman spelar också roll här. Båda kör samma avbildningar och tar emot de flesta av samma kommandon. Docker använder en daemon, som är privilegierad som standard; Podman startar containrar direkt som barnprocesser till kommandot eller till systemd. På en bärbar dator är skillnaden liten. På en server vars brandvägg och processövervakning någon redan har utformat kan den avgöra hela frågan.

Så ser det ut i praktiken

Docker finns med i flera av projekten på den här webbplatsen, och ett av dem är ett medvetet beslut att välja bort det.

Mellan 2022 och 2024 kördes tjugo containeriserade applikationer i Google Cloud och AWS, bland dem analysapplikationer skrivna i R, där det svåra är att återskapa exakt samma uppsättning paket. De tjugo driftsattes och underhölls med enhetlig konfiguration och övervakning, vilket höll dem stabila och reproducerbara och gjorde problem snabbare att diagnostisera i båda molnen. När något gick sönder användes Podman för felsökningen och Kubernetes för att felsöka orkestreringen, så samma avbildningar lästes av tre olika verktyg.

I större skala stod en flotta av hundra Docker-applikationer på värdar med Ubuntu Linux bakom en blandning av Apache och Nginx, och varje applikation behövde ett giltigt TLS-certifikat. Ett ACME-baserat arbetsflöde utfärdade, förnyade och driftsatte certifikaten automatiskt, så att alla hundra behöll giltiga certifikat och det manuella certifikatarbetet försvann.

Det tredje fallet pekar åt andra hållet. På en värddator vars brandväggsregler kommer från en härdningsroll hamnade containermiljön på Podman och Quadlet i stället för Docker, eftersom standardnätverket i Docker skriver om reglerna i kärnans paketfilter och publicerar containerportar ovanför brandväggens egna regler. Containrarna körs i stället som tjänster under systemd och använder värdens nätverk helt utan publicerade portar. Skälet står nedskrivet tillsammans med valet, så nästa person som rör värddatorn vet att standardvalet vägdes och inte glömdes bort.

Var det går fel

Docker gör den första containern enkel. Besväret kommer senare, oftast i en av de här fyra formerna.

En port som brandväggen blockerar svarar från internet. Som standard binds en publicerad port till varje adress på värddatorn, och de paketfilterregler som Docker skriver skickar trafiken till containern innan ett verktyg som ufw ens ser den. Brandväggen rapporterar porten som stängd medan den i själva verket är öppen. När bara en lokal omvänd proxy behöver nå containern, publicera mot loopback-adressen:

```
# reachable from anywhere, whatever ufw reports
docker run -d -p 8080:80 nginx:1.27

# reachable only from the host itself
docker run -d -p 127.0.0.1:8080:80 nginx:1.27
```

Data ligger kvar i containerns eget skrivbara lager. Allt som skrivs dit raderas när containern tas bort, och det är precis vad en uppgradering gör. Databaser, uppladdningar och allt annat som måste överleva hör hemma på en namngiven volym eller en bindmontering, och den volymen behöver i sin tur en egen säkerhetskopia.

Samma Dockerfile bygger något annat än förra veckan. En tagg som `python:3` eller `latest` är en pekare som flyttar sig, så samma Dockerfile kan bygga en annan avbildning en annan dag. Lås basavbildningar till en viss version, eller till en digest när bygget måste gå att upprepa exakt, och bygg om med avsikt när basavbildningen publicerar säkerhetsrättningar.

En inloggningsuppgift följer med i avbildningen. Byggargument och miljövariabler finns kvar i den färdiga avbildningen, och en fil som kopieras in i ett lager stannar i det lagret även om en senare instruktion raderar den. Den som kan hämta avbildningen kan läsa den. Skicka operativa hemligheter till bygget genom en secret mount, som gör dem synliga för ett enda `RUN` -steg utan att skriva in dem i något lager.

Läs mer

- Docker Docs — de officiella manualerna, guiderna och referensen för motorn, kommandoraden och Compose.
- Docker overview — klienten, daemonen, avbildningarna och containrarna förklarade av projektet självt.
- Port publishing and mapping — varför publicerade portar som standard är öppna utåt, och hur de binds till localhost.
- Packet filtering and firewalls — hur Docker skriver regler för iptables eller nftables och var ufw slutar se trafiken.
- Build secrets — hur inloggningsuppgifter når ett bygge utan att lämnas kvar i avbildningen.
- OCI Image Format Specification — den öppna standarden som låter en och samma avbildning köras under Docker, Podman och containerd.

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 Docker?” 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/docker/