Engineer

Docker

en måte å levere et program med alt det trenger på

Hva det er

Med Docker pakker du et program sammen med bibliotekene det bruker, kjøretidsmiljøet for språket og filene det trenger, i én gjenstand som kalles et image, og kjører så det imaget som en isolert prosess, en container. Imaget bygges én gang fra en oppskrift, en Dockerfile, og det samme imaget starter deretter på en bærbar maskin, på en byggserver og i produksjon. Hele løftet får plass i én setning: det som ble testet, er byte for byte det som kjører.

Fire substantiver bærer det meste av ordforrådet. En Dockerfile er en tekstfil med instruksjoner som `FROM`, `COPY` og `RUN`. Et image er en skrivebeskyttet mal bygd fra de instruksjonene og lagret som en stabel med lag, ett per instruksjon, slik at en ny bygging etter en liten endring bare gjør om de lagene som faktisk er endret. En container er en kjørende instans av et image med et tynt, skrivbart lag på toppen. Et register (registry), f.eks. Docker Hub eller en skyleverandørs eget, er stedet der images lastes opp og hentes ned.

Under kommandolinjen ligger en klient og en server. Kommandoen `docker` er klienten. Den sender forespørsler over et REST-API, som regel gjennom en Unix-socket, til en daemon som går hele tiden, `dockerd`, og sammen utgjør de det dokumentasjonen kaller Docker Engine. Daemonen eier images, containere, nettverk og volumer. Selve livssyklusen til containerne gir den videre til containerd, og containerd, som er en containerkjøretid, bruker som standard runc, det lille programmet som ber Linux-kjernen om navnerom (namespaces) og kontrollgrupper (cgroups) og starter prosessen inne i dem. En container er altså en helt vanlig prosess på vertens systemkjerne, med et innsnevret utsyn over resten av systemet. Den er ikke en virtuell maskin, og på macOS og Windows kjører Docker Desktop en liten virtuell Linux-maskin bare for å ha en Linux-kjerne å dele.

Formatet for images vokste fra selskapet som lagde det. I juni 2015 startet Docker, CoreOS og flere andre Open Container Initiative (OCI), og Docker ga bort både formatet sitt og runc til initiativet. Spesifikasjonene som kom ut av det arbeidet, er grunnen til at et image bygd med Docker kjører uendret under Podman, containerd eller en administrert tjeneste i en sky.

Bruk begrepet i en beslutning

Bruk «Hva er Docker?» 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

Spørsmålet som er verdt å stille, er aldri «Docker eller ikke» i det abstrakte. Det er om en bestemt arbeidslast fortjener et image, eller om en vanlig tjeneste på verten ville gjort den samme jobben med færre bevegelige deler.

Et image fortjener plassen sin når programmet drar et helt miljø etter seg. En Python- eller R-applikasjon som trenger ett nøyaktig sett med pakker, en eldre tjeneste som er låst til en gammel kjøretid, to programmer som vil ha hver sin versjon av det samme biblioteket: hvert av dem er tungvint å gjenskape for hånd og trivielt å gjenskape fra et image. Å rulle tilbake betyr å starte det forrige imaget. En ny utvikler kjører én kommando i stedet for å følge en side med notater om oppsettet.

Docker mot en virtuell maskin er den samme avveiningen sett fra den andre siden. Containere starter på et øyeblikk og deler én systemkjerne, så dusinvis får plass på en vert som ville rommet en håndfull virtuelle maskiner. Prisen er svakere isolasjon, siden en sårbarhet i den delte kjernen rammer hver container på verten, mens en virtuell maskin har sin egen.

Kostnaden er ekte infrastruktur. Som standard kjører daemonen som root og styrer nettverket; Docker støtter også rootless-modus. Det følger med et register som noen må drive eller betale for, basisimages som samler opp sikkerhetsrettelser og må bygges på nytt for å få dem, og et lag for datalagring

der reglene er andre enn på verten. En enkelt statisk binærfil uten avhengigheter vinner svært lite på alt dette, og en systemd-enhet i sandkasse kjører den like godt.

Forskjellen mellom Docker og Podman teller også her. Begge kjører de samme imagene og godtar stort sett de samme kommandoene. Docker bruker en daemon, som er privilegert som standard, mens Podman starter containerne direkte som barneprosesser av kommandoen eller av systemd. På en bærbar maskin er forskjellen liten. På en server der noen allerede har utformet brannmuren og styringen av prosessene, kan den avgjøre hele saken.

Slik ser det ut i praksis

Docker går igjen i arbeidet på dette nettstedet, og ett av prosjektene er et bevisst valg om å la det være.

Fra 2022 til 2024 kjørte tjue containeriserte applikasjoner i Google Cloud og AWS, blant dem R-baserte analyseapper, der det vanskelige er å gjenskape nøyaktig det samme settet med pakker. De tjue ble produksjonssatt og vedlikeholdt med konsekvent konfigurasjon og overvåking, slik at ingen av dem ble et spesialtilfelle. Når noe gikk galt, gikk feilsøkingen gjennom Podman og Kubernetes. Applikasjonene holdt seg stabile og reproduserbare, og problemer ble raskere å diagnostisere på tvers av begge skyene.

I større skala sto en flåte på hundre Docker-applikasjoner på verter med Ubuntu Linux, bak en blanding av Apache og Nginx, og hver av dem trengte et gyldig TLS-sertifikat. En ACME-basert arbeidsflyt opprettet, fornyet og distribuerte sertifikatene automatisk, så alle de hundre beholdt gyldige sertifikater, og det manuelle sertifikatarbeidet forsvant.

Det tredje tilfellet peker motsatt vei. På en vert der brannmurreglene installeres av en rolle for herding, gikk containerlaget over på Podman og Quadlet i stedet for Docker, fordi Docker i standardoppsettet skriver om kjernens regler i pakkefilteret og publiserer containerporter over brannmurens egne regler. Containerne bruker i stedet vertens nettverk uten noen publiserte porter, og det fjerner spørsmålet om å gå utenom brannmuren i stedet for å dempe det. Begrunnelsen er skrevet ned sammen med valget, så den neste som rører verten, vet at standardvalget ble veid og ikke glemt.

Hvor det går galt

Docker gjør den første containeren enkel. Problemene kommer senere, og som regel i en av disse fire formene.

En port som brannmuren blokkerer, svarer fra internett. Som standard binder en publisert port seg til alle adressene på verten, og pakkefilterreglene som Docker skriver, sender trafikken videre til containeren før et verktøy som ufw i det hele tatt ser den. Brannmuren melder at porten er stengt, mens den i virkeligheten står åpen. Når det bare er en lokal omvendt proxy som trenger containeren, publiserer du til loopback-adressen:

```
# nåbar fra hvor som helst, uansett hva ufw melder
docker run -d -p 8080:80 nginx:1.27

# nåbar bare fra verten selv
docker run -d -p 127.0.0.1:8080:80 nginx:1.27
```

Data forsvinner sammen med containeren. Alt som skrives til containerens eget skrivbare lag, slettes når containeren fjernes, og det er nøyaktig det en oppgradering gjør. Databaser, opplastinger og alt annet som må overleve, hører hjemme på et navngitt volum eller en bind mount, og det volumet trenger så en egen sikkerhetskopi. Et volum uten sikkerhetskopi er bare en mer langlivet måte å miste de samme dataene på.

En ny bygging gir noe annet enn forrige uke. En tag som `python:3` eller `latest` er en peker som flytter seg, så den samme Dockerfilen kan bygge et annet image en annen dag. Lås basisimages til en bestemt versjon, eller til en digest når byggingen må kunne gjentas helt nøyaktig, og bygg på nytt med vilje når basisimaget publiserer sikkerhetsrettelser.

Påloggingsinformasjon følger med i imaget. Byggeargumenter og miljøvariabler blir værende i det ferdige imaget, og en fil som kopieres inn i ett lag, blir liggende i det laget selv om en senere instruksjon sletter den. Alle som kan hente imaget, kan lese den. Send driftshemmeligheter til byggingen gjennom en secret mount, som gjør dem tilgjengelige for ett enkelt `RUN` -steg uten å skrive dem inn i noe lag.

Les mer

- Docker Docs — de offisielle manualene, veiledningene og referansen for motoren, kommandolinjen og Compose.
- Docker overview — klienten, daemonen, images og containere forklart av prosjektet selv.
- Port publishing and mapping — hvorfor publiserte porter som standard er åpne utover, og hvordan du binder dem til localhost.
- Packet filtering and firewalls — hvordan Docker skriver regler for iptables eller nftables, og hvor ufw slutter å se trafikken.
- Build secrets — påloggingsinformasjon til en bygging uten at den blir liggende igjen i imaget.
- OCI Image Format Specification — den åpne standarden som lar ett og samme image kjøre under Docker, Podman og containerd.

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 Docker?» 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/docker/