Engineer

Docker

en måde at sende et program med alt, hvad det behøver

Hvad det er

Docker er et sæt værktøjer, der pakker et program sammen med dets biblioteker, sprogets runtime og de filer, det skal bruge, til ét samlet artefakt, et image, og derefter kører det image som en isoleret proces: en container. Opskriften er en Dockerfile. Når imaget først er bygget, starter præcis det samme image på udviklerens bærbare, på byggeserveren og i produktion. Det er hele idéen: det, der blev testet, er det, der kører.

Ordforrådet hviler på fire begreber. En Dockerfile er en tekstfil med instruktioner som `FROM`, `COPY` og `RUN`. Et image er en skrivebeskyttet skabelon bygget ud fra de instruktioner og gemt som en stak lag, ét pr. instruktion, så et nyt build efter en lille ændring kun laver de lag om, der faktisk har ændret sig. En container er en kørende instans af et image med et tyndt skrivbart lag ovenpå. Et registry, for eksempel Docker Hub eller en cloududbyders eget, er stedet, hvor images skubbes op og hentes ned.

Bag kommandolinjen står en klient og en server. Kommandoen `docker` er klienten. Den sender forespørgsler over et REST-API, som regel gennem en Unix-socket, til dæmonen `dockerd`, der kører hele tiden og ejer images, containere, netværk og volumes. Selve containerens livsforløb overlader dæmonen til containerd, som i standardopsætningen bruger runc: det lille program, der beder Linux-kernen om namespaces og control groups og starter processen inde i dem. En container er altså en almindelig proces på værtens kerne med et begrænset udsyn over systemet og ikke en virtuel maskine. Derfor kører Docker Desktop på macOS og Windows en lille virtuel Linux-maskine i baggrunden, så der er en Linux-kerne at dele.

Formatet er vokset fra firmaet, der lavede det. I juni 2015 grundlagde Docker, CoreOS og andre Open Container Initiative, og Docker overdrog sit imageformat og runc til projektet. De specifikationer, der kom ud af det, er grunden til, at et image bygget med Docker kører uændret under Podman, containerd eller en administreret cloudtjeneste.

Brug begrebet i en beslutning

Brug »Hvad er Docker?« 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

Det interessante spørgsmål handler sjældent om Docker i almindelighed. Det handler om, hvorvidt netop denne opgave fortjener et image, eller om en almindelig tjeneste direkte på værten klarer det samme med færre bevægelige dele.

Et image tjener sig hjem, når programmet slæber et helt miljø efter sig. En Python- eller R-applikation, der kræver præcis én samling pakker, en ældre tjeneste låst til en gammel runtime, to programmer, der vil have hver sin version af det samme bibliotek: alt det er besværligt at genskabe i hånden og ligetil at genskabe fra et image. En tilbagerulning består i at starte det forrige image, og en ny udvikler kører én kommando i stedet for at følge en side med opsætningsnoter. Docker over for en virtuel maskine er den samme afvejning set fra den anden side. Containere starter på et øjeblik og deler én kerne, så der er plads til mange flere af dem på den samme vært, mod en svagere isolation.

Prisen er rigtig infrastruktur. Som standard kører dæmonen som root og styrer netværket; Docker understøtter også rootless-tilstand. Der skal drives eller betales for et registry. Basisimages samler sikkerhedsrettelser op og skal bygges igen for at få dem med, og lagringen følger andre regler end værtens filsystem. Et statisk binært program uden afhængigheder får næsten intet ud af alt dette, og en systemd-unit med sandkasse kører det lige så godt.

Forskellen mellem Docker og Podman hører også hjemme her. De kører de samme images og forstår de fleste af de samme kommandoer, men Docker bruger en dæmon, som er privilegeret som standard, mens Podman starter containerne direkte som børn af kommandoen eller af systemd. På en bærbar betyder det ikke ret meget. På en server, hvor nogen allerede har gennemtænkt firewall og procesovervågning, kan det afgøre sagen.

Sådan ser det ud i praksis

Docker går igen i arbejdet på dette site, og et af projekterne handler om et bevidst fravalg.

Fra 2022 til 2024 kørte tyve containeriserede applikationer fordelt på Google Cloud og AWS, heriblandt analyseapplikationer skrevet i R, hvor den svære del netop er at ramme den rigtige pakkesamling hver gang. De tyve blev udrullet og vedligeholdt med ensartet konfiguration og overvågning, og det holdt dem stabile og reproducerbare og gjorde fejlsøgningen hurtigere på tværs af de to skyer.

Hundrede Docker-applikationer på Ubuntu-værter bag en blanding af Apache og Nginx skulle hver have et gyldigt TLS-certifikat. Et ACME-baseret workflow udstedte, fornyede og udrullede certifikaterne automatisk, så alle hundrede beholdt gyldige certifikater, og det manuelle certifikatarbejde forsvandt.

Det tredje eksempel går den modsatte vej. På en vært, hvis firewallregler kommer fra en hærdningsrolle, blev containerplanet bygget på Podman og Quadlet i stedet for Docker, fordi Dockers standardnetværk omskriver kernens pakkefilterregler og publicerer containerporte over firewallens egne regler. Begrundelsen står skrevet ved beslutningen, så den næste, der rører værten, kan se, at standardvalget blev vejet og ikke glemt.

Hvor det går galt

Den første container er nem at få op. Besværet kommer senere og tager som regel en af disse fire former.

En port, som firewallen spærrer, svarer fra internettet. Som standard binder en publiceret port til alle værtens adresser, og de pakkefilterregler, Docker skriver, sender trafikken videre til containeren, før et værktøj som ufw

overhovedet ser den. Firewallen viser porten som lukket, mens den står åben. Skal kun en lokal reverse proxy nå containeren, så publicér porten på loopback-adressen:

```
# kan nås udefra, uanset hvad ufw viser
docker run -d -p 8080:80 nginx:1.27

# kan kun nås fra værten selv
docker run -d -p 127.0.0.1:8080:80 nginx:1.27
```

Data forsvinder sammen med containeren. Alt, hvad der skrives til containerens eget skrivbare lag, slettes, når containeren fjernes, og det er præcis, hvad en opgradering gør. Databaser, uploadede filer og alt andet, der skal overleve, hører hjemme på en navngiven volume eller en bind mount, og den volume skal så have sin egen backup.

Et nyt build giver noget andet end i sidste uge. Et tag som `python:3` eller `latest` er en pegepind, der flytter sig, så den samme Dockerfile kan bygge et andet image en anden dag. Lås basisimages til en bestemt version, eller til et digest, når buildet skal kunne gentages præcist, og byg bevidst igen, når basisimaget får sikkerhedsrettelser.

Adgangsoplysninger bliver sendt ud med imaget. Build-argumenter og miljøvariabler bliver liggende i det færdige image, og en fil, der kopieres ind i ét lag, bliver i det lag, selv om en senere instruktion sletter den. Alle, der kan hente imaget, kan læse den. Giv i stedet hemmeligheder til buildet gennem en secret mount, som kun gør dem synlige for et enkelt `RUN` -trin og aldrig skriver dem ind i et lag.

Læs mere

- Docker Docs — de officielle manualer, vejledninger og opslag for motoren, kommandolinjen og Compose.
- Docker overview — klienten, dæmonen, images og containere forklaret af projektet selv.
- Port publishing and mapping — hvorfor publicerede porte som standard er åbne udadtil, og hvordan de bindes til localhost.
- Packet filtering and firewalls — hvordan Docker skriver regler i iptables eller nftables, og hvor ufw mister trafikken af syne.
- Build secrets — hemmeligheder til et build uden at efterlade dem i imaget.

- OCI Image Format Specification — den åbne standard, der lader ét image køre under Docker, Podman og containerd.

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 Docker?« 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/docker/