Engineer

Produksjonssetting

å flytte en endring dit folk faktisk bruker den

Hva det er

Produksjonssetting er rekken av steg som tar en testet versjon av en applikasjon ut av byggesystemet og setter den i drift. De bygde filene eller containerimaget legges på maskinene eller plattformen som skal kjøre dem, versjonen får konfigurasjonen for miljøet sitt, den startes, og ekte forespørsler flyttes over til den. Mange norske team sier bare deploy eller deployment, og verbet blir «å deploye» eller «å sette i produksjon». Navnet sier produksjon, men ordet brukes like gjerne når målet er staging eller et testmiljø.

Ordet betegner både handlingen og resultatet. Et team snakker om en produksjonssetting som stoppet ved databasesteget, og om produksjonssettingene som kjører i hvert miljø akkurat nå. Det som settes i produksjon, kan være et nettsted, et API, en planlagt jobb, en endring i en database eller backenden bak en mobilapp.

Produksjonssetting og utgivelse holdes ofte fra hverandre. Produksjonssettingen legger den nye koden der den kan kjøre; utgivelsen avgjør hvem som får se den. En funksjon skjult bak et flagg kan settes i produksjon mandag og gis ut til kundene torsdag, uten at noen rører serverne igjen. Skillet lar et team levere kode i små, hyppige steg, mens beslutningen om når kundene får noe, blir liggende hos dem som eier produktet.

Mange produksjonssettinger leverer i dag et containerimage. Open Container Initiative, OCI, ble opprettet under Linux Foundation i juni 2015 og gir ut tre spesifikasjoner rundt det: hvordan et image er bygd opp, hvordan en containerkjøretid kjører det når det er pakket ut, og hvordan registre lagrer og deler ut image. Et image identifiseres ved sin digest, en hash av innholdet,

så samme digest betyr alltid de samme bytene. En tag som `1.4.2` er en lesbar peker til et image, og en ny push under samme tag får den til å peke på noe annet.

The Twelve-Factor App, en mye sitert metodikk for å bygge webtjenester, deler veien fra kode til kjørende tjeneste i tre faser som holdes strengt fra hverandre:

- **Build** gjør en commit om til en kjørbar pakke.
- **Release** kombinerer pakken med konfigurasjonen for ett miljø.
- **Run** starter prosesser fra en valgt utgivelse.

Hver utgivelse får en unik identifikator og endres aldri etter at den er laget. Enhver endring gir en ny utgivelse, og det er det som gjør tilbakerulling til et spørsmål om å velge en tidligere.

Hvordan selve byttet skjer, avhenger av hva tjenesten tåler:

- **Recreate** stopper den gamle versjonen og starter så den nye, og godtar et hull imellom.
- **Rullerende utrulling** bytter ut instanser noen få om gangen, så begge versjonene betjener trafikk en stund. Det er standarden for et Deployment i Kubernetes.
- **Blue-green** kjører den nye versjonen ved siden av den gamle og flytter all trafikk på én gang, med den gamle klar til å ta den tilbake.
- **Canary** sender først en liten andel av trafikken til den nye versjonen og øker andelen så lenge feilrater og latens holder seg friske.

Bruk begrepet i en beslutning

Bruk «Hva er produksjonssetting?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Testet, utgitt og mulig å reversere, slik at endringer når kundene en tirsdag ettermiddag i stedet for sent en fredagskveld.

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. Automatiserte GIS-SaaS-utrulling i GitHub Actions — produksjonssetting, data og rapportering, alt uten manuelle grep

Automatiserte GIS-SaaS-utrulling i GitHub Actions Ta kontakt →

Hvorfor det er viktig

Produksjonssettingen er punktet der en endring begynner å hjelpe noen, og også punktet der en feil begynner å nå brukerne. Team som setter i produksjon sjelden, samler gjerne mange endringer i hver utgivelse. Da bærer hver produksjonssetting mer risiko, og når den ryker, er det flere kandidater til årsaken. Små og hyppige produksjonssettinger holder hvert steg lett å forstå og lett å reversere.

En produksjonssetting for hånd er bare så konsistent som personen som gjør den. Steg hoppes over, tas i en annen rekkefølge eller gjøres litt annerledes av en kollega som dekker en ferie. En leveransepipeline gjør de samme stegene hver gang og etterlater et spor: hvilken commit, hvilken artefakt og hvilket miljø, med tidsstempel. Det sporet er der gjennomgangen etter en driftshendelse begynner, og ofte den raskeste veien til svaret på hva som ble endret.

Den sterkeste vanen er å bygge én gang og promotere resultatet. Artefakten som ble testet i staging, skal være byte for byte den samme som når produksjon, og bare konfigurasjonen skal skille dem. Miljøet leverer den konfigurasjonen når artefakten kommer fram. Når digesten registreres ved byggingen og hver produksjonssetting skjer etter digest, blir løftet mulig å etterprøve:

```
podman build -t registry.example.com/web:1.4.2 .
podman push --digestfile web.digest registry.example.com/web:1.4.2
kubectl -n staging set image deployment/web \
  web=registry.example.com/web@$(cat web.digest)
# når kontrollene i staging er bestått, går samme digest til produksjon
kubectl -n production set image deployment/web \
  web=registry.example.com/web@$(cat web.digest)
```

Tilbakerulling hviler på den samme disiplinen. Når utgivelser er uforanderlige og har en identifikator, betyr det å gå tilbake å peke på den forrige, og det tar sekunder. Når hvert miljø bygger på nytt fra kildekoden, betyr det å bygge en gammel commit på nytt og håpe at de samme avhengighetsversjonene fortsatt velges.

Slik ser det ut i praksis

Produksjonssettingen av én plattform til Azure ble eid fra ende til annen, og hver endring gikk den samme faste ruten fra utvikling til staging til produksjon, uten at noen pushet rett inn i et miljø i drift. Leveransepipelinen bygger imagene og ruller dem ut, mens Bicep-malene holder målinfrastrukturen lik fra miljø til miljø, så hver promotering lander på det samme oppsettet. Innstillinger som varierer mellom miljøene, holdes atskilt fra hemmelighetene. Det lar ett og samme bygg gå gjennom alle trinnene og plukke opp innstillingene for hvert miljø der det lander, uten å bygges på nytt underveis. Siden melder at endringer nådde hvert miljø langs den definerte veien, uten ad hoc-produksjonssettinger for hånd, og at utgivelser ble et kontrollert steg som kjører likt hver gang.

Et tidligere oppdrag viser hva den konsistensen er verdt i timer. Automatiserte leveransepipeliner for bygging, testing og produksjonssetting sparte omtrent 4 000 timer ved å standardisere veien fra en endring til produksjon og fjerne manuelle steg som varierte med hvem som utførte dem. Siden melder at utgivelsene ble både raskere og mer pålitelige.

På selskapets egen plattform holdes produksjonssetting atskilt fra provisjonering. Et containerlag bygd på Podman og Quadlet lar root sette opp stien for produksjonssetting én gang. Deretter setter en uprivilegert bruker i produksjon på nytt ved å kjøre ett fast skript gjennom en snevert avgrenset regel som tillater den ene kommandoen uten argumenter. Skriptet bygger imaget og starter tjenesten på nytt, så en rutinemessig utgivelse trenger verken root-shell eller vilkårlige kommandoer.

Hvor det går galt

Det meste av trøbbel med produksjonssetting kommer av en forskjell mellom det som ble sjekket, og det som faktisk endret seg.

Produksjon kjører noe staging aldri så. Når hvert miljø bygger sin egen kopi, velges nyere versjoner innenfor avhengighetenes versjonsintervaller, et baseimage flytter seg mellom byggene, eller et byggeflagg er annerledes, og artefakten som besto testingen, er en annen enn den som betjener kundene. Bygg én gang, registrer digesten, promoter den digesten, og hold alt som er miljøspesifikt, i konfigurasjonen og ikke i artefakten.

En tag flytter seg, og en tilbakerulling lander på feil versjon. En produksjonssetting som peker på `latest`, eller som gjenbruker en versjonstag etter et nytt bygg, avhenger av hva taggen peker på den dagen den kjører. Kubernetes-dokumentasjonen fraråder `:latest` i produksjon, fordi det blir vanskeligere å se hvilken versjon som kjører, og vanskeligere å rulle tilbake ordentlig. Bruk en versjonstag som aldri gjenbrukes, og lås digesten der plattformen tillater det.

Pipelinen melder suksess før den nye versjonen virker. Mange kommandoer for produksjonssetting returnerer så snart plattformen har tatt imot forespørselen, mens de nye instansene fortsatt starter og kan krasje et minutt senere. Et steg som venter til den nye versjonen melder seg klar, og så sender noen ekte forespørsler og sjekker svarene, gjør en stille svikt om til et rødt bygg mens den forrige utgivelsen fortsatt er like for hånden.

Ny kode og gammelt databaseskjema møtes midt i en utrulling. Under en rullerende utrulling eller en canary betjener gamle og nye versjoner trafikk side om side mot én database. En databasemigrering som gir en kolonne nytt navn eller fjerner den, ødelegger de gamle instansene som fortsatt kjører, og gjør tilbakerulling utrygg, fordi den gamle koden forventer det som ble fjernet. Del slike endringer over flere produksjonssettinger: legg til den nye strukturen først, lever kode som virker med begge, og fjern den gamle strukturen først når ingen kjørende versjon trenger den.

Les mer

- Build, release, run — argumentet i The Twelve-Factor App for tre atskilte faser og utgivelser som aldri endres når de først er laget.
- Open Container Initiative — prosjektet i Linux Foundation bak spesifikasjonene for image, kjøretid og distribusjon som de fleste containerbaserte produksjonssettinger bygger på.
- OCI Image Format Specification — manifester, lag, konfigurasjon og innholdsdigestene som identifiserer et image.
- Kubernetes Deployments — rullerende oppdateringer, recreate-strategien, fristen for fremdrift og tilbakerulling.
- Kubernetes images — tagger, digester, pull-policyer og hvorfor `:latest` gjør tilbakerulling vanskeligere.

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 produksjonssetting?» 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/deployment/