Engineer

Deployment

at flytte en ændring derhen hvor folk faktisk bruger den

Hvad det er

Deployment er de trin, der tager en testet version af en applikation ud af byggesystemet og sætter den i drift: det byggede image eller de byggede filer lægges på de maskiner eller den platform, der skal køre dem, versionen får konfigurationen til netop det miljø, den startes, og rigtige forespørgsler flyttes over til den. På dansk bruges både det engelske ord og udrulning. Ordet dækker både handlingen og resultatet, så man kan tale om en deployment, der gik i stå ved databasetrinnet, og om de deployments, der kører i hvert miljø lige nu. Det, der rulles ud, kan være et website, et API, et planlagt job, en databaseændring eller backenden bag en mobilapp.

At deploye og at frigive er ofte to forskellige ting. At deploye lægger den nye kode der, hvor den kan køre; at frigive afgør, hvem der får den at se. En funktion bag et feature flag kan deployes om mandagen og slås til for kunderne om torsdagen, uden at nogen rører serverne igen. Adskillelsen lader et team sende kode ud i små, hyppige skridt, mens beslutningen om timing bliver hos dem, der ejer produktet.

Mange udrulninger i dag sender et containerimage af sted. Open Container Initiative blev oprettet i juni 2015 under Linux Foundation og udgiver tre specifikationer omkring det: hvordan et image er bygget op, hvordan en runtime kører det, når det er pakket ud, og hvordan registries gemmer og udleverer images. Et image identificeres ved sit digest, en hashværdi af indholdet, så det samme digest altid betyder de samme bytes. Et tag som `1.4.2` er en læsbar henvisning til et image, og pusher man igen under samme tag, peger det på noget nyt.

Metoden The Twelve-Factor App, som mange henviser til, når talen falder på webtjenester, deler vejen fra kode til kørende tjeneste i tre trin, der holdes skarpt adskilt. Byggetrinnet gør et commit til en eksekverbar pakke. Udgivelsestrinnet kombinerer pakken med ét miljøs konfiguration. Kørselstrinnet starter processer fra en valgt udgivelse. Hver udgivelse får sit eget id og ændres aldrig, når den først findes; enhver ændring giver en ny udgivelse, og derfor handler en tilbagerulning blot om at vælge en tidligere.

Selve skiftet kan ske på flere måder, alt efter hvad tjenesten kan tåle. Ved recreate stoppes den gamle version, før den nye startes, og man accepterer et hul imellem. En rullende udrulning skifter instanser ud nogle få ad gangen, så begge versioner betjener trafik et stykke tid; det er standarden for et Kubernetes Deployment. Ved blue-green kører den nye version ved siden af den gamle, al trafik flyttes på én gang, og den gamle står klar til at tage over igen. En canary sender først en lille del af trafikken til den nye version og udvider, så længe fejlrate og svartider holder sig pæne.

Brug begrebet i en beslutning

Brug »Hvad er deployment?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Testet, udgivet og til at rulle tilbage, så ændringer når kunderne en tirsdag eftermiddag i stedet for sent en fredag aften.

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 automatiseret GIS-SaaS-udrulning med CI/CD — udrulning, data og rapportering, alt sammen uden manuelt arbejde

Har automatiseret GIS-SaaS-udrulning med CI/CD Kontakt os →

Hvorfor det betyder noget

Det er ved udrulningen, at en ændring begynder at gøre gavn, og det er også her, en fejl begynder at nå brugerne. Teams, der sjældent deployer, samler gerne mange ændringer i hver udgivelse, så hver udrulning bærer mere risiko og, når den går galt, flere mulige syndere. Små, hyppige udrulninger holder hvert skridt let at overskue og let at fortryde.

En udrulning i hånden er kun så ensartet som den, der udfører den. Trin springes over, tages i en anden rækkefølge eller gøres lidt anderledes af kollegaen, der dækker en ferie. En udrulningspipeline tager de samme trin hver gang og efterlader et spor: hvilket commit, hvilket artefakt, hvilket miljø og hvornår. Det spor er stedet, hvor en gennemgang af en hændelse begynder, og ofte den hurtigste vej til at svare på, hvad der ændrede sig.

Den vigtigste vane er at bygge én gang og forfremme resultatet. Artefaktet, der blev testet i staging, skal være præcis det samme, som når produktion, og kun konfigurationen må variere; den leverer miljøet, når artefaktet ankommer. Gemmer man digestet ved bygningen og deployer efter digest, kan løftet efterprøves:

```
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@$(cat web.digest)
# samme digest går til produktion, når staging er godkendt
kubectl -n production set image deployment/web \
  web=registry.example.com/web@$(cat web.digest)
```

Tilbagerulning hviler på samme disciplin. Når udgivelser er uforanderlige og har et id, betyder det at gå tilbage blot at pege på den forrige, og det tager sekunder. Når hvert miljø bygger fra kildekoden, betyder det at bygge et gammelt commit igen og håbe, at de samme afhængighedsversioner stadig vælges.

Sådan ser det ud i praksis

Udrulningen af én platform til Azure blev ejet fra start til slut, og hver ændring tog den samme vej gennem udvikling, staging og produktion, uden at nogen skubbede direkte ud i et levende miljø. Udrulningspipelinen bygger images og sender dem ud, og Bicep-skabeloner holder målinfrastrukturen ens i alle miljøer, så hver forfremmelse lander på matchende infrastruktur. Indstillinger, der varierer fra miljø til miljø, holdes adskilt fra hemmelighederne, og derfor kan ét build rejse gennem alle trin og samle hvert miljøs indstillinger op ved ankomsten uden at blive bygget om undervejs. Siden fortæller, at ændringer når hvert miljø ad den faste vej uden manuelle ad hoc-deploys, og at udgivelser blev et kontrolleret trin, der forløber ens hver gang.

En tidligere opgave viser, hvad den slags ensartethed er værd i timer. Automatisering af bygge-, test- og udrulningspipelines sparede omkring 4.000 timer ved at gøre vejen fra ændring til produktion til en standard og fjerne manuelle trin, der afhang af, hvem der udførte dem. Siden melder, at udgivelser blev både hurtigere og mere pålidelige.

På virksomhedens egen platform holdes udrulning adskilt fra provisionering. Et containerplan bygget på Podman og Quadlet lader root forberede udrulningsvejen én gang. Derefter er hver udrulning ét fast script, som en uprivilegeret bruger kalder gennem en snævert afgrænset regel, der kun tillader den kommando uden argumenter. Scriptet bygger imaget og genstarter tjenesten, så en almindelig udgivelse hverken kræver en root-shell eller vilkårlige kommandoer.

Hvor det går galt

De fleste problemer med udrulning opstår i forskellen mellem det, der blev tjekket, og det, der faktisk ændrede sig.

Produktion kører noget, staging aldrig har set. Når hvert miljø bygger sin egen kopi, vælges nyere versioner af afhængighederne, et base-image flytter sig mellem to builds, eller et byggeflag er sat anderledes, og artefaktet, der bestod testen, er et andet end det, kunderne rammer. Byg én gang, gem digestet, forfrem det digest, og læg alt miljøspecifikt i konfigurationen frem for i artefaktet.

Et tag flytter sig, og en tilbagerulning lander på den forkerte version. En udrulning, der nævner `latest` eller genbruger et versionstag efter et nyt build, afhænger af, hvad tagget peger på den dag, den kører. Kubernetes-dokumentationen fraråder `:latest` i produktion, fordi det bliver sværere at se, hvilken version der kører, og sværere at rulle ordentligt tilbage. Brug versionstags, der aldrig genbruges, og lås digestet fast, hvor platformen tillader det.

Pipelinen melder succes, før den nye version virker. Mange deploy-kommandoer vender tilbage, så snart platformen har modtaget anmodningen, mens de nye instanser stadig starter og måske går ned et minut senere. Et trin, der venter på, at den nye version melder klar, og derefter sender nogle få rigtige forespørgsler og tjekker svarene, gør en stille fejl til et rødt build, mens den forrige udgivelse stadig er lige ved hånden.

Ny kode og det gamle skema mødes midt i en udrulning. Under en rullende udrulning eller en canary betjener gamle og nye versioner trafik side om side mod én database. En migrering, der omdøber eller sletter en kolonne, vælter de gamle instanser, der stadig kører, og gør tilbagerulning usikker, fordi den gamle

kode forventer det, der blev fjernet. Fordel den slags ændringer over flere udrulninger: tilføj den nye struktur først, send kode ud, der virker med begge, og fjern først den gamle struktur, når ingen kørende version har brug for den.

Læs mere

- Build, release, run — Twelve-Factor-metodens argument for tre adskilte trin og udgivelser, der aldrig ændres, når de først er lavet.
- Open Container Initiative — projektet under Linux Foundation bag specifikationerne for images, runtimes og distribution, som de fleste containerudrulninger hviler på.
- OCI Image Format Specification — manifester, lag, konfiguration og de digests, der identificerer et image.
- Kubernetes Deployments — rullende opdateringer, recreate-strategien, fremdriftsfristen og tilbagerulning.
- Kubernetes images — tags, digests, pull-politikker og hvorfor `:latest` gør tilbagerulning sværere.

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 deployment?« 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/deployment/