Infrastrcuture as Code

Infrastructure Code Testing

Forfattere:
Tor Ivar Melling

Eiere:
Forfatterne og Forskningsstiftelsen TISIP

Innhold

1. INTRODUKSJON

I moderne IT-miljøer handler ikke infrastruktur lenger bare om å bygge servere eller sette opp nettverk. Det handler om å levere og endre infrastruktur på en trygg, rask og forutsigbar måte. Kief Morris beskriver dette som *Core Infrastructure Delivery Workflows* – arbeidsflytene som gjør det mulig å behandle infrastruktur som kode og levere den med samme kvalitet og kontroll som programvare.

Kjernen i denne tilnærmingen er kontinuerlig levering. I stedet for å gjøre store og risikable endringer i sjeldne omganger, bygges og testes små oppdateringer fortløpende. Hver endring blir versjonskontrollert, testet automatisk og levert gjennom en standardisert prosess. Dette gir både stabilitet og fleksibilitet: systemene kan utvikles raskt uten at kvaliteten går tapt.

Morris legger vekt på at automatisering er en forutsetning for god *Infrastructure as Code*. Hele leveringsprosessen, fra utvikling til drift, bør være automatisert. Det reduserer menneskelige feil, øker repeterbarheten og frigjør tid til forbedring og innovasjon. Automatiseringen må imidlertid være strukturert og forståelig. Derfor er det viktig å etablere klare stadier i leveringsprosessen – typisk utvikling, bygging, testing, utrulling og drift – og å sørge for at kode, prosesser og verktøy henger sammen på tvers av disse stadiene.

Kapittel 17 og 18 i boken danner grunnlaget for en dypere forståelse av hvordan denne prosessen ser ut i praksis. Kapittel 17, *Infrastructure Code Testing Strategy*, forklarer hvordan testing av infrastrukturkode skal inngå som en naturlig del av leveringsflyten. Her ser Morris til prinsipper fra smidig utvikling og test-drevet utvikling (TDD), og viser hvordan de samme prinsippene kan brukes for å bygge tillit til infrastrukturen som kode. Kapittel 18, *Infrastructure Code Testing Implementation*, viser hvordan dette faktisk kan implementeres ved hjelp av ulike testnivåer, fra raske syntakskontroller til omfattende systemtester som kjører i skyen.

2. CONTINUOUS DELIVERY PRINCIPLES FOR INFRASTRUCTURE AS CODE

Kontinuerlig levering, ofte forkortet *CD*, er kjernen i moderne utviklingsarbeid. For infrastruktur betyr det å kunne gjøre små, trygge og repeterbare endringer som raskt kan tas i bruk uten at det går på bekostning av stabiliteten. Kief Morris beskriver dette som en forlengelse av prinsippene fra programvareutvikling, men anvendt på selve grunnmuren systemene hviler på.

I stedet for at infrastrukturen bygges manuelt, som en engangsoperasjon, behandles den som levende kode. Hver endring går gjennom en kjede av trinn som sikrer kvalitet og stabilitet. Når en utvikler foreslår en oppdatering i Terraform-konfigurasjonen, lagres den først i versjonskontrollsystemet – vanligvis Git. Derfra utløser endringen en automatisert prosess: koden bygges, valideres og testes før den eventuelt blir godkjent for utrulling.

Dette betyr at infrastrukturendringer håndteres med samme disiplin som programvareendringer. Endringer kan spores, rulles tilbake, gjennomgås i *pull requests* og verifiseres i testmiljøer før de påvirker produksjonen. Morris understreker at denne arbeidsflyten gir trygghet, men også tempo. Når teamene vet at prosessen vil fange opp feil tidlig, kan de levere endringer raskere og mer selvsikkert.

Et viktig prinsipp i kontinuerlig levering er automatisering. Enhver manuell handling representerer en potensiell feilkilde. Derfor bør bygging, testing og utrulling av infrastruktur være helautomatisert. Det gjelder alt fra å kjøre terraform validate og terraform plan til å teste driftsmiljøet etter at endringer er utført. Ved å fjerne menneskelig variasjon sikres det at infrastrukturen kan reproduseres nøyaktig – hver gang.

Et annet sentralt poeng Morris trekker frem, er behovet for *små, hyppige endringer*. I tradisjonell drift ble infrastrukturen ofte oppdatert i store omganger, noe som gjorde prosessen både risikabel og tungvint. I en IaC-tilnærming leveres endringer i små biter, der hver bit testes og godkjennes. Dermed blir det lettere å forstå konsekvensene av en endring, og enklere å feilsøke hvis noe går galt.

Til slutt påpeker Morris at kontinuerlig levering ikke bare handler om verktøy og prosesser, men også om kultur. Det krever at utviklere og driftsansvarlige samarbeider tett, deler ansvar og har et felles mål: å bygge og vedlikeholde en stabil og fleksibel plattform. Når kultur, kode og automatisering virker sammen, oppstår en infrastruktur som både er robust og tilpasningsdyktig – en forutsetning for moderne DevOps.

3. PIPELINE STAGES FOR INFRASTRUCTURE DELIVERY

Når infrastrukturen behandles som kode, må den leveres gjennom en forutsigbar og standardisert prosess. Kief Morris beskriver denne prosessen som en *pipeline* – en serie trinn som hver sørger for at infrastrukturen er korrekt, testet og klar for utrulling. Målet er å oppnå høy kvalitet uten å bremse hastigheten i utviklingen.

En slik pipeline kan sammenlignes med en fabrikk: råmaterialet – infrastrukturen som kode – går inn i den ene enden, og et ferdig, driftssatt miljø kommer ut i den andre. Underveis blir koden bygget, testet, kontrollert og validert. Hvert steg i prosessen har sin egen rolle, og det er nettopp denne strukturen som gjør det mulig å levere komplekse miljøer trygt og repeterbart.

Den første fasen er build, hvor infrastrukturen klargjøres for testing og utrulling. I denne fasen blir koden validert – syntaksen kontrolleres, og verktøy som *TFLint* og *Checkov* kan brukes for å avdekke feil eller avvik fra beste praksis. Dette sikrer at koden er korrekt og følger organisasjonens retningslinjer før den i det hele tatt vurderes for distribusjon.

Deretter følger test-fasen, hvor infrastrukturen faktisk prøves ut. Her kan man sette opp et midlertidig miljø, ofte kalt et *staging environment*, hvor Terraform-planen kjøres og resultatet evalueres. Morris fremhever at denne fasen bør være helautomatisert, slik at testmiljøet alltid er en nøyaktig kopi av produksjonsmiljøet. På denne måten kan feil oppdages tidlig, før de får konsekvenser i drift.

Neste steg er deployment, der infrastrukturen rulles ut til produksjon. Morris understreker at denne fasen skal være så automatisert som mulig – helst helt uten manuell inngripen. En god pipeline sørger for at bare testet og godkjent kode får slippe gjennom til produksjon. Endringen skal være både sporbar og reverserbar, slik at teamet kan rulle tilbake hvis noe går galt.

Til slutt beskriver Morris drifts- og overvåkningsfasen, som ofte blir undervurdert i IaC-arbeid. Selv etter at infrastrukturen er levert, må den overvåkes kontinuerlig for å oppdage avvik, ytelsesproblemer og *drift drift* – altså forskjeller mellom infrastrukturen slik den er beskrevet i koden, og slik den faktisk kjører. Automatiserte kontroller og rapporter er derfor avgjørende for å opprettholde konsistens og pålitelighet over tid.

Morris legger også vekt på at pipeline-stegene ikke må sees som rigide faser, men som en kontinuerlig syklus. Hver endring som gjøres – enten det er en liten oppdatering i en modul eller en større justering av et miljø – går gjennom de samme fasene på nytt. Dette gjør det mulig å forbedre infrastrukturen trinnvis, og samtidig beholde full kontroll.

4. TESTING STRATEGY FOR INFRASTRUCTURE CODE

Testing er en hjørnestein i arbeidet med Infrastructure as Code. Kief Morris fremhever at infrastruktur, på samme måte som programvare, må testes systematisk for å kunne stoles på. Målet er å bygge tillit til at infrastrukturen fungerer slik koden beskriver, og at endringer ikke skaper uforutsette konsekvenser.

En god teststrategi for IaC handler derfor ikke bare om å finne feil, men om å skape trygghet. Når infrastrukturen leveres automatisk gjennom en pipeline, må man vite at hver endring er kontrollert og forutsigbar. Testing sørger for nettopp dette – den gjør leveransen pålitelig.

Morris skiller mellom flere testnivåer, som sammen dekker hele livssyklusen til infrastrukturen. Hvert nivå har et bestemt formål, og de utfyller hverandre.

Det første nivået er unit testing, som retter seg mot de minste byggesteinene i infrastrukturen – for eksempel en modul eller en variabel i Terraform. Her tester man at modulene fungerer isolert, uten å faktisk opprette ressurser i skyen. Verktøy som *terraform validate* eller *Terratest* kan brukes for å sjekke at koden oppfører seg som forventet. Unit tester gir rask tilbakemelding og fanger opp mange småfeil tidlig.

Neste nivå er integration testing, der man ser på samspillet mellom flere moduler eller komponenter. En virtuell maskin kan for eksempel trenge nettverkstilgang, lagring og sikkerhetsregler. Disse delene må fungere sammen for at infrastrukturen skal virke i praksis. Her kan testmiljøet settes opp automatisk, og ressursene opprettes midlertidig for å verifisere at samhandlingen fungerer.

Deretter kommer system testing, hvor man tester hele infrastrukturen slik den ville vært i produksjon. Dette nivået krever et mer fullstendig testmiljø og kan inkludere ytelsestesting, lasttesting og feilscenarier. Selv om slike tester tar lengre tid, er de avgjørende for å avdekke problemer som bare oppstår når alt settes sammen.

Det siste nivået Morris trekker frem, er compliance testing. Her testes infrastrukturen mot krav til sikkerhet, kostnad og standarder for organisasjonen. Dette kan inkludere kontroller av tilgangsregler, kryptering, nettverksgrenser eller naming conventions. Verktøy som *Checkov* og *InSpec* kan brukes for å kjøre slike automatiserte kontroller. På denne måten sikres det at all infrastruktur oppfyller selskapets krav – uten manuelle sjekker.

Morris understreker at testingen må være kontinuerlig. Det holder ikke å teste koden én gang før utrulling. I stedet bør testene kjøres automatisk hver gang en endring foreslås, hver gang koden

merges, og regelmessig etter at infrastrukturen er i drift. På den måten oppdages avvik og feil så tidlig som mulig.

En effektiv teststrategi handler altså om å bygge tillit, ikke bare kode. Ved å teste på flere nivåer, kombinert med automatisering og gode prosesser, blir infrastrukturen både robust og fleksibel. Når utviklere kan stole på at testene fanger opp feil, tør de å gjøre endringer oftere – og det er nettopp slik organisasjoner oppnår høy endringstakt uten å miste kontroll.

Til sammen skaper disse trinnene en rytme i leveransen: valider, test, lever, overvåk, og lær. Det er denne rytmen som gjør at et IaC-prosjekt kan bevege seg raskt uten å miste stabilitet, og som gjør pipeline-baserte arbeidsflyter til selve ryggraden i moderne infrastrukturleveranser.

5. IMPLEMENTING INFRASTRUCTURE CODE TESTING

Etter at prinsippene og nivåene for testing er på plass, handler neste steg om å sette teorien ut i livet. Kief Morris viser hvordan en teststrategi blir effektiv først når den bygges inn i den daglige arbeidsflyten. Testing må være en naturlig del av infrastrukturen – ikke noe som gjøres som et ekstra steg på slutten.

I praksis betyr det at testene skal være en integrert del av *pipeline*-en. Når en utvikler foreslår en endring i koden, skal hele kjeden av tester kjøre automatisk – fra validering til systemtesting. Hensikten er å sikre at feil blir fanget opp før de rekker å påvirke produksjon.

Morris understreker at testene må være raske og repeterbare. I tidlige faser, som ved *pull requests*, skal testene gi umiddelbar tilbakemelding. Unit- og syntakstester bør derfor kjøre på sekunder, ikke minutter. De mer omfattende testene, som integrasjon og system, kan kjøre i parallelle miljøer, slik at utviklingsflyten ikke bremses.

For å få dette til i praksis foreslår Morris en strukturert oppbygging av testmiljøene. Et typisk mønster er å ha flere miljøer som speiler hverandre:

- Et dev- eller feature-miljø, hvor nye endringer testes isolert.
- Et staging-miljø, som speiler produksjonen og brukes til integrasjonstesting.
- Et produksjonsmiljø, hvor kun godkjent kode får rulles ut.

Disse miljøene bygges og oppdateres automatisk med IaC-verktøy som Terraform, og testene knyttes direkte til pipeline-trinnene. For eksempel kan en GitHub Actions-workflow først kjøre terraform fmt og terraform validate, deretter bruke *TFLint* for kvalitetskontroll og *Terratest* for å kjøre tester mot et midlertidig miljø. Dersom testene består, kan koden automatisk fremmes til staging.

Morris legger også stor vekt på opprydding. Når midlertidige testmiljøer er brukt opp, må de slettes igjen. Automatisert opprydding hindrer ressurs-sløsing og holder testmiljøene konsistente. Han viser til at dette kan bygges direkte inn i CI/CD-pipelinen – for eksempel ved å kjøre terraform destroy etter at testene er fullført.

Et annet viktig aspekt i implementeringen er testdata og determinisme. For at testene skal være pålitelige, må de gi samme resultat hver gang. Infrastruktur som testes må derfor være forutsigbar og fri for tilfeldige variasjoner. Dette betyr at data, navn og parametere i testene må kontrolleres nøye.

Morris minner også om at automatiserte tester ikke kan erstatte menneskelig vurdering, men de kan redusere risiko og øke tempoet. Manuelle gjennomganger og *peer reviews* av koden bør fortsatt være en del av prosessen – særlig i større eller mer sensitive endringer. Samspillet mellom automatiske og manuelle kontroller gir det beste grunnlaget for kvalitet.

Til slutt fremhever Morris at kontinuerlig forbedring er en del av testimplementeringen. Etter hvert som teamet lærer mer om feil, driftsproblemer eller svakheter i infrastrukturen, bør nye tester legges til. En testpipeline er aldri ferdig – den utvikler seg i takt med systemet den beskytter.

Testing som en del av leveringsløpet handler dermed om mer enn bare verktøy. Det handler om å bygge en kultur der hver endring blir testet, validert og forstått. Når testene er godt integrert i pipeline og utviklingsrutiner, blir de et naturlig sikkerhetsnett – og fundamentet for trygg, rask og forutsigbar infrastrukturlevering.

6. FROM TESTING TO CONTINUOUS DELIVERY – BUILDING THE CORE WORKFLOW

Når teststrategien og implementeringen smelter sammen, oppstår det som Morris kaller et *Core Infrastructure Delivery Workflow*. Dette er mer enn bare en teknisk prosess – det er et helhetlig rammeverk for hvordan organisasjoner kan bygge, endre og levere infrastruktur trygt og effektivt over tid.

Et slikt arbeidsløp starter alltid med kode. All infrastruktur beskrives eksplisitt som kode i et versjonskontrollsystem, der hver endring er synlig, sporbar og reverserbar. Når en utvikler gjør en endring, for eksempel i en Terraform-modul, utløses en pipeline som automatisk validerer, tester og deployer endringen. Denne automatiserte kjeden sørger for at kvaliteten på infrastrukturen holdes høy, uavhengig av hvor mange ganger den endres.

Testing utgjør selve hjertet i denne prosessen. Som Morris viser, er testing ikke bare et kontrollsteg, men en kontinuerlig aktivitet som følger koden fra idé til drift. Små, raske tester sikrer kvalitet tidlig, mens større integrasjons- og systemtester bekrefter at infrastrukturen fungerer helhetlig. Resultatet er en infrastruktur som stadig forbedres uten å miste stabilitet.

Et godt *Core Infrastructure Delivery Workflow* er bygget opp av tre nøkkelprinsipper. Det første er automatisering, som fjerner manuelle feil og skaper repeterbarhet. Det andre er standardisering, som gjør at alle endringer følger samme prosess – uansett hvem som gjør dem. Det tredje er kontinuerlig tilbakemelding, som gir utviklere og driftsteam rask innsikt i hvordan endringene påvirker miljøet.

Morris fremhever også at kultur og samarbeid er avgjørende. Et automatisert leveringsløp har liten verdi hvis ikke organisasjonen deler ansvar for kvaliteten på infrastrukturen. DevOps-tankegangen handler derfor om å bryte ned barrierer mellom utvikling og drift. Alle som arbeider med systemet, skal bidra til at infrastrukturen er testbar, trygg og lett å vedlikeholde.

Når testingen er på plass i alle ledd – fra kode til produksjon – blir infrastrukturen både robust og smidig. Nye funksjoner kan leveres raskt, feil kan oppdages tidlig, og driftsstabiliteten øker. Samtidig får teamene selvtillit til å eksperimentere og forbedre, fordi de vet at endringer kan reverseres og kontrolleres.

Til slutt minner Morris om at et slikt arbeidsløp aldri er ferdig. Infrastruktur, verktøy og behov endrer seg kontinuerlig. Derfor må også prosessene utvikles videre. Nye tester legges til, pipelines forbedres, og arbeidsmetodene justeres i takt med erfaring og læring. Det er nettopp denne kontinuerlige forbedringen som gjør Infrastructure as Code til mer enn bare et verktøy –

det er en tilnærming til hvordan organisasjoner tenker, samarbeider og bygger sine digitale systemer.