Engineer

Infrastruktur som kode

å styre infrastruktur fra versjonerte definisjonsfiler

Hva det er

Infrastruktur som kode er praksisen med å beskrive det et system kjører på (maskiner, nettverk, brannmurregler, lastbalanserere, DNS-oppføringer) i tekstfiler, holde filene i versjonskontroll og la et verktøy bygge det virkelige miljøet ut fra dem. Filene er originalen. En endring i infrastrukturen begynner som en endring i en fil, går gjennom den samme gjennomgangen som applikasjonskoden og når det kjørende systemet først når verktøyet utfører den. På engelsk heter det infrastructure as code, og forkortelsen IaC brukes på begge språk.

Veiledningen NIST SP 800-204C, som handler om DevSecOps for programvare bygd av mikrotjenester, definerer infrastruktur som kode slik: kode som oppretter og konfigurerer ressursene et program rulles ut på, gjentakbart og ensartet hver gang. Ifølge den samme veiledningen versjonsstyres definisjonene med de samme verktøyene som programmets egen kildekode. Gjentakbarheten er hele poenget. Et miljø som kan lages fra filer, kan lages igjen: ved siden av det første, etter et havari eller i en testkonto en tirsdag ettermiddag.

Definisjoner finnes i to hovedformer. En deklarativ definisjon beskriver sluttresultatet, for eksempel en server av en gitt størrelse i et gitt nettverk med én port åpen, og lar verktøyet finne ut hva som skal opprettes, endres eller få være i fred. En imperativ definisjon er en liste med steg: opprett dette, koble til det, og åpne så porten. SP 800-204C plasserer infrastruktur som kode under deklarativ kode: teamet skriver ned hva det trenger, og et verktøy gjør det om til en kjørende instans. HashiCorp beskriver Terraform-språket på samme måte: det oppgir et mål, ikke stegene som fører dit. Forskjellen viser seg ved andre kjøring. En liste med steg forteller hvordan noe ble gjort én gang, mens en deklarasjon forteller hva som skal finnes, og det er nettopp det verktøyet må vite når en del av det allerede er der.

Det finnes ikke ett felles språk. SP 800-204C påpeker at koden, avhengig av verktøyet, skrives enten i et vanlig skriptspråk som Python, JavaScript eller TypeScript, eller i verktøyets eget konfigurasjonsspråk, for eksempel HashiCorps HCL, som kan være forenlig med et standardformat som JSON, men ikke trenger å være det. Terraforms språk hører til den siste sorten og ligger i filer som slutter på `.tf`. Azure tar imot Resource Manager-maler i JSON eller i det kortere språket Bicep. Ansible skriver playbookene sine i YAML og når maskinene over SSH med deres eksisterende påloggingsopplysninger, uten at det installeres en agent på dem.

Det hjelper å skille mellom to jobber som begge går under navnet. Klargjøring oppretter delene: den virtuelle maskinen, disken, nettverket, brannmuren. Konfigurasjon former det som kjører inne i en maskin som allerede finnes: pakker, kontoer, tjenestefiler. Noen verktøy heller mot den ene jobben og noen mot den andre, og grensen er flytende. Det som gjør begge til infrastruktur som kode, er at definisjonen ligger i et repo, og at verktøyet bringer det virkelige systemet i samsvar med den.

Verktøyene husker også ulikt hva de har bygd. Terraform fører en tilstandsfil som kobler ressursene i konfigurasjonen til de virkelige objektene det har opprettet, og dokumentasjonen kaller den tilstanden sannhetskilden for miljøet. Bicep har ingen tilstandsfil, fordi Azure selv registrerer hva som er rullet ut. Hvor oversikten ligger, avgjør hva som skjer når to personer kjører verktøyet samtidig, eller når oversikten går tapt.

Bruk begrepet i en beslutning

Bruk «Hva er infrastruktur som kode?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Hele oppsettet skrevet ned som filer, slik at det kan gjenskapes nøyaktig og aldri bare finnes i hodet på én person.

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. Skrev 1 000 RBAC-regler for å sikre Linux-servere — 1,000 tilgangsregler skrevet for å tette et sikkerhetshull

Skrev 1 000 RBAC-regler for å sikre Linux-servere Ta kontakt →

Hvorfor det er viktig

Før infrastruktur som kode ble hvert miljø satt opp for hånd og siden lappet for hånd. Testmiljøet fikk en rettelse som produksjon ikke fikk, en port åpnet for en test ble aldri lukket, og én maskin av tre fikk en nyere pakke. SP 800-204C bruker det engelske ordet drift om dette, i betydningen at noe glir ut av kurs:

konfigurasjonen som er i bruk, gir ikke lenger den tilstanden systemet var ment å være i. Veiledningen peker også på den vanlige veien inn, en endring gjort i en nettkonsoll eller på kommandolinjen utenom koden. Når en release virker i testmiljøet og feiler i produksjon, er det ofte denne typen avvik som viser seg.

Infrastruktur som kode flytter originalen ut av det kjørende systemet. Teamet endrer definisjonen, og det kjørende systemet følger etter. Ingen logger inn for å rette en brannmurregel; man retter filen som beskriver brannmuren, og verktøyet utfører endringen. Det gir tre ting.

Den første er gjennomgang og historikk. En endring i nettverket blir en diff i en pull request som en annen person leser før noe skjer. Måneder senere har spørsmålet om hvem som åpnet en port, og hvorfor, et svar i commit-loggen og ikke i noens hukommelse. Å angre en endring er en revert fulgt av en ny kjøring av verktøyet.

Den andre er repeterbarhet. En velskrevet definisjon er idempotent: brukt på et miljø som allerede stemmer, endrer den ingenting, og brukt på et som har glidd unna, bringer den det tilbake. Det finnes to veier dit: enten justeres det som står der, eller det kastes og et nytt bygges opp fra definisjonen. Den andre veien, gjort til en fast regel, er immutable infrastructure.

Den tredje er billige miljøer. SP 800-204C beskriver verktøyene som automatisering av hele infrastrukturens levetid, fra klargjøring til nedleggelse, og påpeker at noe av den lever så kort at det ikke lønner seg å vedlikeholde den. Et team kan opprette flere testmiljøer ved behov og rive dem ned igjen, og selve koden kan sjekkes før den rører noe: veiledningen krever at hver del av den skannes for sårbarheter før den merges. En gren kan få sin egen kortlevde kopi av produksjonsoppsettet, og en som gjennomgår endringen, kan se en forhåndsvisning av den før den godkjennes.

Kostnadene bør også sies rett ut. Å endre én brannmurregel via en pull request går tregere enn å klikke i en nettkonsoll, og under et avbrudd føles det dyrt. Hvert verktøy har sitt eget språk og sin egen modell for tilstand som må læres. Og en fil som kan bygge et miljø, kan også slette det: spaken rekker like langt begge veier, og derfor betyr forhåndsvisningen like mye som kjøringen etterpå.

Slik ser det ut i praksis

Her er én endring på vei fra redigering til kjørende system, i et oppsett der en CI-jobb kjører verktøyet og legger tørrkjøringen på pull requesten:

```
Dag 1   09.10  Rediger nettverksdefinisjonen: tillat TCP 443 fra kontoret
        09.12  Åpne en pull request; CI gjør en tørrkjøring (planen)
        09.14  Planen ligger på pull requesten: én regel lagt til, ingen fjernet
        10.30  En kollega leser planen i tillegg til diffen, og godkjenner
        10.31  Merge; CI utfører planen som ble godkjent
        10.33  En ny kjøring av samme definisjon melder ingen endringer
Dag 40          Noen spør hvorfor 443 er åpen: commit-loggen sier hvem og hvorfor
```

Planen er den delen som betaler for seg. Diffen viser hva forfatteren ville endre; planen viser hva verktøyet faktisk skal gjøre, også det forfatteren ikke regnet med, for eksempel en ressurs det har tenkt å fjerne. Terraform kan lagre en plan i en fil med `-out`, og `terraform apply` utfører da operasjonene i akkurat den filen, så det som kjører kl. 10.31, er det kollegaen godkjente kl. 10.30. Den stille kjøringen kl. 10.33 er kontrollen av at definisjonen og systemet nå er enige.

Her på nettstedet beskriver tjenestesiden om infrastruktur som kode gjentakbar klargjøring av servere fra versjonsstyrt konfigurasjon i stedet for å være avhengig av noens hukommelse.

Å komme dit trygt er mest et spørsmål om rekkefølge, og en porteføljeside fra september 2026 viser hvorfor. Den handler om en ekstra server hos en annen leverandør. Den forrige maskinen var satt opp for hånd og sikret senere, så en nyopprettet vert med bare en standard passordpolicy en stund var åpen mot internett. Denne gangen ble serveren opprettet fra repoet, og beskyttelsen var på plass før maskinen var det. Brannmur-playet kjøres før server-playet, og hos den andre leverandøren tildeles taggene før skybrannmuren, siden regler rettet mot tagger ikke har noe å feste seg til før taggene er der. Å kjøre på nytt skader ingenting, for hvis maskinen allerede finnes, tar rollen den i bruk i stedet for å opprette en til. Den rollen er den eneste som koster penger, så dokumentasjonen dens oppgir regionen, maskintypen med spesifikasjon og prisen per måned i euro. Siden er også ærlig om hva en tørrkjøring går glipp av. Begge brannmurene konfigureres ved å kalle leverandørenes REST-API-er fra en generell HTTP-oppgave, og den oppgaven støtter ikke check mode. Derfor hopper en tørrkjøring over den. Den egentlige øvingen er en kjøring av brannmur-playet alene, som ikke koster noe og kan rulles tilbake.

Hvor det går galt

Infrastrukturkode feiler gjerne på måter som ligner suksess. Verktøyet kjører ferdig uten feil, CI-kjøringen er grønn, og filene og det virkelige systemet er stille og rolig uenige.

En rask rettelse under en hendelse blir fjernet av neste kjøring. Noen åpner en port i konsollen klokka to om natta, avbruddet er over, og en uke senere lukker den faste kjøringen porten igjen, fordi filene aldri fikk vite om den. Det motsatte skjer også: ingen kjører verktøyet igjen, og filene slutter å beskrive virkeligheten. Skriv nødrettelsen inn i koden samme dag, og kjør en planlagt sjekk som sammenligner filene med det som kjører; SP 800-204C ber om den sammenligningen rett etter hver utrulling og med jevne mellomrom mens systemet kjører. Med Terraform gjør den detaljerte avslutningskoden til plan-kommandoen det til en kort nattlig jobb:

```
terraform plan -detailed-exitcode
# exit 0: ingen endringer, exit 1: feil, exit 2: endringer funnet
```

På en gren som allerede er utført, betyr avslutningskode 2 at det virkelige systemet har beveget seg bort fra koden, og at noen bør finne ut hvorfor.

Å slette en ressurs fra filen sletter den ikke alltid. Verktøyene er uenige om hva en manglende ressurs betyr. For Azure Resource Manager-maler skriver Microsoft i dokumentasjonen til what-if at en ressurs som finnes, men ikke står i malen, bare slettes i complete mode. I andre moduser kjører den videre og koster fortsatt penger. Lær verktøyets regel for fjerning før du stoler på den, og les forhåndsvisningen for slettingene du venter deg, i tillegg til dem du frykter.

Tørrkjøringen ser ikke alt. En forhåndsvisning er en modell av endringen, ikke en garanti. Microsofts what-if-dokumentasjon nevner egenskaper som meldes som slettet selv om Azure bare fyller inn en standardverdi under utrullingen, og kaller resultatet støy; enhver egenskap som settes via `reference` -funksjonen, vises som endret ved hver kjøring. Porteføljesiden over viser det motsatte hullet: en oppgave uten støtte for check mode hoppes over i en tørrkjøring, så forhåndsvisningen sier ingenting om den. Kjenn den kjente støyen i verktøyet ditt, og regn alt forhåndsvisningen ikke ser, som utestet til en øvelse på noe billig og reversibelt har dekket det.

Hemmeligheter havner i repoet eller i tilstanden. Et passord skrevet inn i en definisjonsfil blir liggende i historikken i hver eneste klone. Mindre opplagt: Terraforms dokumentasjon advarer om at tilstanden kan inneholde hemmeligheter, og som standard er den tilstanden en lokal fil som heter `terraform.tfstate`. Hold hemmeligheter i et hemmelighetslager og vis til dem ved navn, legg tilstanden i fjernlagring med låsing og tilgangskontroll, slik HashiCorp anbefaler, og rediger aldri tilstandsfilen for hånd; kommandoen `terraform state` finnes for de endringene som trengs.

Et skript forkledd som definisjon. Et skallskript i repoet som oppretter en server, er under versjonskontroll, men det er ikke idempotent: kjør det to ganger, og det står to servere, eller en feil midt i det hele. Bruk deklarative ressurser eller steg som sjekker før de handler, slik som rollen over som tar over en eksisterende maskin. Ansibles dokumentasjon setter listen tydelig: når et system allerede er i tilstanden en playbook beskriver, endrer en ny kjøring av playbooken ingenting.

Les mer

- Implementation of DevSecOps for a Microservices-based Application with Service Mesh (NIST SP 800-204C) — deklarativ, versjonsstyrt infrastrukturkode, avvik og skanning.
- Terraform Language Documentation — det deklarative språket, ressursene og blokkene i det.
- Command: plan — tørrkjøringen, lagrede planer og de detaljerte avslutningskodene.
- State — hva Terraform registrerer, hvor det ligger, og hvorfor det er sensitivt.
- Introduction to Ansible — agentløs automatisering over SSH, playbooks og ønsket tilstand.
- Template deployment what-if — forhåndsvisning av en Azure-utrulling, endringstypene og den kjente støyen.

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 infrastruktur som kode?» 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/infrastructure-as-code/