Engineer

Infrastructure as code

at styre infrastruktur fra versionerede definitionsfiler

Hvad det er

Infrastructure as code er praksis med at beskrive det, et system kører på (maskiner, netværk, firewallregler, loadbalancere, DNS-poster), i tekstfiler, holde filerne i versionsstyring og lade et værktøj bygge det rigtige miljø ud fra dem. Filerne er originalen. En ændring af infrastrukturen begynder som en ændring i en fil, gennemgår samme review som programkoden og når først det kørende system, når værktøjet udfører den. På dansk skrives det også infrastruktur som kode, og forkortelsen IaC går igen på begge sprog.

Vejledningen NIST SP 800-204C, der handler om DevSecOps for software bygget af microservices, beskriver infrastructure as code som den kode, der opretter og konfigurerer de ressourcer, et program rulles ud på, og gør det gentageligt og ensartet hver gang. Ifølge samme vejledning versionsstyres definitionerne med de samme værktøjer som programmets egen kildekode. Gentageligheden er hele pointen. Et miljø, der kan skabes ud fra filer, kan skabes igen: ved siden af det første, efter et nedbrud eller i en testkonto en tirsdag eftermiddag.

Definitioner findes i to hovedformer. En deklarativ definition beskriver slutresultatet, for eksempel en server af en bestemt størrelse i et bestemt netværk med én port åben, og overlader til værktøjet at regne ud, hvad der skal oprettes, ændres eller lades i fred. En imperativ definition er en liste af trin: opret dette, tilknyt hint, og åbn så porten. SP 800-204C regner infrastructure as code til deklarativ kode: teamet skriver ned, hvad det har brug for, og et værktøj gør det til en kørende instans. HashiCorp beskriver Terraform-sproget på samme måde: det angiver et mål og ikke de trin, der fører derhen. Forskellen viser sig ved anden kørsel. En liste af trin fortæller, hvordan noget blev gjort én gang, mens en erklæring fortæller, hvad der skal findes, og det er netop det, værktøjet skal vide, når en del af det allerede er der.

Der findes ikke ét fælles sprog. SP 800-204C nævner, at koden alt efter værktøjet skrives enten i et almindeligt scriptsprog som Python, JavaScript eller TypeScript eller i værktøjets eget konfigurationssprog, for eksempel HashiCorps HCL, som kan være forenelig med et standardformat som JSON, men ikke behøver at være det. Terraforms sprog hører til den sidste slags og ligger i filer, der ender på `.tf`. Azure tager imod Resource Manager-skabeloner i JSON eller i det kortere sprog Bicep. Ansible skriver sine playbooks i YAML og når maskinerne over SSH med deres eksisterende brugeroplysninger, uden at der installeres en agent på dem.

Det hjælper at skelne mellem to opgaver, der begge går under navnet. Provisionering skaber delene: den virtuelle maskine, disken, netværket, firewallen. Konfiguration former det, der kører inde i en maskine, som allerede findes: pakker, konti, servicefiler. Nogle værktøjer hælder mod den ene opgave og nogle mod den anden, og grænsen er flydende. Det, der gør begge dele til infrastructure as code, er, at definitionen ligger i et repository, og at værktøjet bringer det rigtige system i overensstemmelse med den.

Værktøjerne husker også forskelligt, hvad de har bygget. Terraform fører en tilstandsfil, der kobler ressourcerne i konfigurationen til de rigtige objekter, det har oprettet, og dokumentationen kalder den tilstand sandhedskilden for miljøet. Bicep har ingen tilstandsfil, fordi Azure selv registrerer, hvad der er rullet ud. Hvor optegnelsen ligger, afgør, hvad der sker, når to personer kører værktøjet samtidig, eller når optegnelsen går tabt.

Brug begrebet i en beslutning

Brug »Hvad er infrastructure as code?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Hele opsætningen skrevet ned som filer, så den kan genskabes præcist og aldrig kun findes i én persons hoved.

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 skrevet 1.000 RBAC-regler til Linux-servere — 1,000 adgangsregler skrevet for at lukke et sikkerhedshul

Har skrevet 1.000 RBAC-regler til Linux-servere Kontakt os →

Hvorfor det betyder noget

Før infrastructure as code blev hvert miljø sat op i hånden og siden lappet i hånden. Staging fik en rettelse, som produktion ikke fik, en port åbnet til en test blev aldrig lukket, og én maskine ud af tre fik en nyere pakke. SP 800-204C kalder fænomenet drift, i betydningen at noget glider væk: opsætningen i brug giver

ikke længere den tilstand, systemet var tænkt til at være i. Vejledningen peger også på den sædvanlige vej derind, en ændring lavet i en webkonsol eller på kommandolinjen uden om koden. Når en release virker i staging og fejler i produktion, er det ofte den slags afvigelse, der viser sig.

Infrastructure as code flytter originalen ud af det kørende system. Teamet ændrer definitionen, og det kørende system følger efter. Ingen logger ind for at rette en firewallregel; man retter filen, der beskriver firewallen, og værktøjet udfører ændringen. Det giver tre ting.

Den første er review og historik. En ændring af netværket bliver til en diff i en pull request, som en anden person læser, før noget sker. Måneder senere har spørgsmålet om, hvem der åbnede en port og hvorfor, et svar i commit-loggen og ikke i nogens hukommelse. At fortryde en ændring er et revert efterfulgt af en ny kørsel af værktøjet.

Den anden er gentagelighed. En velskrevet definition er idempotent: anvendt på et miljø, der allerede passer, ændrer den ingenting, og anvendt på et, der er gledet væk, bringer den det tilbage. Der er to veje dertil: enten justeres det, der står, eller det kasseres, og et nyt bygges op fra definitionen. Den anden vej, gjort til en fast regel, er immutable infrastructure.

Den tredje er billige miljøer. SP 800-204C beskriver værktøjerne som automatisering af infrastrukturens hele levetid, fra provisionering til nedlæggelse, og bemærker, at noget af den lever så kort, at det ikke kan betale sig at vedligeholde den. Et team kan oprette flere testmiljøer efter behov og rive dem ned igen, og selve koden kan tjekkes, før den rører noget: vejledningen kræver, at hver eneste del af den scannes for sårbarheder, før den merges. En gren kan få sin egen kortlivede kopi af produktionens opbygning, og en reviewer kan se en forhåndsvisning af en ændring, før den godkendes.

Omkostningerne skal også siges lige ud. At ændre én firewallregel via en pull request er langsommere end at klikke i en webkonsol, og under et nedbrud føles det dyrt. Hvert værktøj har sit eget sprog og sin egen model for tilstand, som skal læres. Og en fil, der kan bygge et miljø, kan også slette det: håndtaget rækker lige langt begge veje, og derfor betyder forhåndsvisningen lige så meget som kørslen bagefter.

Sådan ser det ud i praksis

Her er én ændring på vej fra redigering til kørende system, i et setup hvor et CI-job kører værktøjet og lægger prøvekørslen på pull requesten:

```
Dag 1   09.10  Ret netværksdefinitionen: tillad TCP 443 fra kontoret
        09.12  Åbn en pull request; CI laver en prøvekørsel (planen)
        09.14  Planen står på pull requesten: én regel tilføjet, ingen fjernet
        10.30  En reviewer læser planen såvel som diffen og godkender
        10.31  Merge; CI udfører den plan, der blev godkendt
        10.33  En ny kørsel af samme definition melder ingen ændringer
Dag 40         Nogen spørger, hvorfor 443 er åben: commit-loggen siger hvem
```

Planen er den del, der tjener sig ind. Diffen viser, hvad forfatteren ville ændre; planen viser, hvad værktøjet faktisk vil gøre, også det forfatteren ikke regnede med, for eksempel en ressource, det har tænkt sig at fjerne. Terraform kan gemme en plan i en fil med `-out`, og `terraform apply` udfører så handlingerne i netop den fil, så det, der kører kl. 10.31, er det, revieweren godkendte kl. 10.30. Den stille kørsel kl. 10.33 er kontrollen af, at definitionen og systemet nu er enige.

Her på sitet beskriver ydelsessiden om Infrastructure as Code gentagelig provisionering af servere ud fra versionsstyret konfiguration i stedet for at være afhængig af nogens hukommelse.

Det sikre forløb handler mest om rækkefølge, og en porteføljeside fra september 2026 viser hvorfor. Den beskriver en ekstra server, som skulle stå hos en anden udbyder. Den tidligere maskine var sat op i hånden og først beskyttet senere, så en nyoprettet vært, der ikke havde andet at stå imod med end en standardpolitik for adgangskoder, var en tid tilgængelig fra internettet. Denne gang blev serveren oprettet fra repositoryet, og beskyttelsen var på plads, før maskinen var. Firewall-playet køres forud for server-playet, og hos den anden udbyder tildeles taggene før cloud-firewallen, da regler rettet mod tags ikke har noget at hæfte sig på, før taggene er der. En ny kørsel gør ingen skade, for findes maskinen allerede, tager rollen den blot i brug i stedet for at lave en til. Den rolle er den eneste, der koster penge, så dens egen dokumentation angiver regionen, maskintypen med specifikation og prisen pr. måned i euro. Siden er også ærlig om, hvad en prøvekørsel ikke fanger. Begge firewalls konfigureres ved at kalde udbydernes REST-API'er fra en generel HTTP-opgave, og den opgave

understøtter ikke check-tilstand. Derfor lader en prøvekørsel den ligge. Den egentlige generalprøve er en kørsel af firewall-playet alene, som intet koster og kan fortrydes.

Hvor det går galt

Infrastrukturkode fejler gerne på måder, der ligner succes. Værktøjet kører færdigt uden fejl, CI-kørslen er grøn, og filerne og det rigtige system er stille og roligt uenige.

En hurtig rettelse under en hændelse bliver fjernet af næste kørsel. Nogen åbner en port i konsollen klokken to om natten, nedbruddet er overstået, og en uge senere lukker den faste kørsel porten igen, fordi filerne aldrig hørte om den. Det modsatte sker også: ingen kører værktøjet igen, og filerne holder op med at beskrive virkeligheden. Skriv nødrettelsen ind i koden samme dag, og kør et planlagt tjek, der sammenligner filerne med det, der kører; SP 800-204C beder om den sammenligning lige efter hver udrulning og med jævne mellemrum, mens systemet kører. Med Terraform gør plan-kommandoens detaljerede exitkode det til et kort natligt job:

```
terraform plan -detailed-exitcode
# exit 0: ingen ændringer, exit 1: fejl, exit 2: ændringer fundet
```

På en gren, der allerede er udført, betyder exitkode 2, at det rigtige system har bevæget sig væk fra koden, og at nogen bør finde ud af hvorfor.

At slette en ressource fra filen sletter den ikke altid. Værktøjerne er uenige om, hvad en manglende ressource betyder. For Azure Resource Manager-skabeloner skriver Microsoft i dokumentationen til what-if, at en ressource, der findes, men ikke står i skabelonen, kun slettes i complete mode. I andre tilstande kører den videre og koster fortsat penge. Lær dit værktøjs regel for fjernelse, før du stoler på den, og læs forhåndsvisningen for de sletninger, du forventer, såvel som dem, du frygter.

Prøvekørslen ser ikke alt. En forhåndsvisning er en model af ændringen, ikke en garanti. Microsofts what-if-dokumentation nævner egenskaber, der meldes som slettet, selv om Azure blot udfylder en standardværdi under udrulningen, og kalder resultatet støj; enhver egenskab, der sættes via `reference` -funktionen, vises som ændret ved hver kørsel. Porteføljesiden ovenfor viser det modsatte

hul: en opgave uden understøttelse af check-tilstand springes over i en prøvekørsel, så forhåndsvisningen siger intet om den. Kend dit værktøjs kendte støj, og betragt alt, hvad forhåndsvisningen ikke kan se, som utestet, indtil en afprøvning på noget billigt, der kan rulles tilbage, har dækket det.

Hemmeligheder havner i repositoryet eller i tilstanden. En adgangskode skrevet ind i en definitionsfil bliver liggende i historikken i hver eneste klon. Mindre oplagt: Terraforms dokumentation advarer om, at tilstanden kan indeholde hemmeligheder, og som standard er den tilstand en lokal fil ved navn `terraform.tfstate`. Hold hemmeligheder i et hemmelighedslager og henvis til dem ved navn, læg tilstanden i fjernlager med låsning og adgangskontrol, som HashiCorp anbefaler, og ret aldrig tilstandsfilen i hånden; kommandoen `terraform state` findes til de ændringer, der er brug for.

Et script forklædt som definition. Et shellscript i repositoryet, der opretter en server, er versionsstyret, men det er ikke idempotent: kør det to gange, og der står to servere, eller en fejl midt i det hele. Brug deklarative ressourcer eller trin, der tjekker, før de handler, som rollen ovenfor, der overtager en eksisterende maskine. Ansibles dokumentation sætter barren klart: når et system allerede er i den tilstand, en playbook beskriver, ændrer en ny kørsel af playbooken ingenting.

Læs mere

- Implementation of DevSecOps for a Microservices-based Application with Service Mesh (NIST SP 800-204C) — deklarativ, versionsstyret infrastrukturkode, afvigelser og scanning.
- Terraform Language Documentation — det deklarative sprog, dets ressourcer og dets blokke.
- Command: plan — prøvekørslen, gemte planer og de detaljerede exitkoder.
- State — hvad Terraform registrerer, hvor det ligger, og hvorfor det er følsomt.
- Introduction to Ansible — agentløs automatisering over SSH, playbooks og ønsket tilstand.
- Template deployment what-if — forhåndsvisning af en Azure-udrulning, ændringstyperne og den kendte støj.

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 infrastructure as code?« 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/infrastructure-as-code/