Engineer

Infrastruktur som kod

att styra infrastruktur från versionerade definitionsfiler

Vad det är

Infrastruktur som kod är arbetssättet att beskriva det som ett system körs på (maskiner, nätverk, brandväggsregler, lastbalanserare, DNS-poster) i textfiler, hålla filerna under versionshantering och låta ett verktyg bygga den verkliga miljön utifrån dem. Filerna är originalet. En ändring av infrastrukturen börjar som en ändring i en fil, granskas på samma sätt som applikationskoden och når det körande systemet först när verktyget genomför den. På engelska heter det infrastructure as code, och förkortningen IaC används på båda språken.

Vägledningen NIST SP 800-204C, som handlar om DevSecOps för programvara byggd av mikrotjänster, definierar infrastruktur som kod så här: kod som skapar och konfigurerar de resurser som ett program rullas ut på, upprepbart och enhetligt varje gång. Enligt samma vägledning versionshanteras definitionerna med samma verktyg som programmets egen källkod. Upprepbarheten är hela poängen. En miljö som kan skapas ur filer kan skapas igen: bredvid den första, efter ett haveri eller i ett testkonto en tisdagseftermiddag.

Definitioner finns i två huvudformer. En deklarativ definition beskriver slutresultatet, till exempel en server av en viss storlek i ett visst nätverk med en port öppen, och låter verktyget räkna ut vad som ska skapas, ändras eller lämnas orört. En imperativ definition är en lista med steg: skapa det här, koppla in det där och öppna sedan porten. SP 800-204C räknar infrastruktur som kod till deklarativ kod: teamet skriver ned vad det behöver, och ett verktyg gör om det till en körande instans. HashiCorp beskriver Terraforms språk på samma sätt: det anger ett mål, inte stegen dit. Skillnaden syns vid andra körningen. En lista med steg berättar hur något gjordes en gång, medan en deklaration berättar vad som ska finnas, och det är just det som verktyget behöver veta när en del av det redan finns.

Det finns inget gemensamt språk. SP 800-204C påpekar att koden, beroende på verktyget, skrivs antingen i ett vanligt skriptspråk som Python, JavaScript eller TypeScript eller i verktygets eget konfigurationsspråk, till exempel HashiCorps HCL, som kan vara förenligt med ett standardformat som JSON men inte behöver vara det. Terraforms språk hör till det senare slaget och ligger i filer som slutar på `.tf`. Azure tar emot Resource Manager-mallar i JSON eller i det kortare språket Bicep. Ansible skriver sina playbooks i YAML och når maskinerna över SSH med deras befintliga inloggningsuppgifter, utan att någon agent installeras på dem.

Det hjälper att skilja på två jobb som båda ryms under namnet. Provisionering skapar delarna: den virtuella maskinen, disken, nätverket, brandväggen. Konfiguration formar det som körs inuti en maskin som redan finns: paket, konton, tjänstefiler. Vissa verktyg lutar åt det ena jobbet och andra åt det andra, och gränsen är flytande. Det som gör båda till infrastruktur som kod är att definitionen ligger i ett repo och att verktyget för det verkliga systemet i överensstämmelse med den.

Verktygen minns också på olika sätt vad de har byggt. Terraform för en tillståndsfil som kopplar resurserna i konfigurationen till de verkliga objekt det har skapat, och dokumentationen kallar det tillståndet för miljöns sanningskälla. Bicep har ingen tillståndsfil, eftersom Azure självt registrerar vad som har rullats ut. Var registret ligger avgör vad som händer när två personer kör verktyget samtidigt, eller när registret går förlorat.

Använd begreppet i ett beslut

Använd ”Vad är infrastruktur som kod?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Hela uppsättningen nedskriven som filer, så att den kan återskapas exakt och aldrig bara finns i en enda persons huvud.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Skrev 1 000 RBAC-regler som säkrar Linux-servrar — 1,000 åtkomstregler, skrivna för att täppa till en säkerhetslucka

Skrev 1 000 RBAC-regler som säkrar Linux-servrar Hör av er →

Varför det spelar roll

Före infrastruktur som kod sattes varje miljö upp för hand och lappades sedan för hand. Testmiljön fick en rättelse som produktionen inte fick, en port som öppnades för ett test stängdes aldrig och en maskin av tre fick ett nyare paket. SP 800-204C använder det engelska ordet drift om detta, i betydelsen att något

glider ur kurs: konfigurationen som används ger inte längre det tillstånd som systemet var tänkt att vara i. Vägledningen pekar också ut den vanliga vägen in, en ändring som görs i en webbkonsol eller på kommandoraden förbi koden. När en release fungerar i testmiljön och fallerar i produktion är det ofta den sortens avvikelse som visar sig.

Infrastruktur som kod flyttar originalet ut ur det körande systemet. Teamet ändrar definitionen, och det körande systemet följer efter. Ingen loggar in för att rätta en brandväggsregel; man rättar filen som beskriver brandväggen, och verktyget genomför ändringen. Det ger tre saker.

Den första är granskning och historik. En ändring i nätverket blir en diff i en pull request som en andra person läser innan något händer. Månader senare har frågan om vem som öppnade en port, och varför, ett svar i commit-loggen i stället för i någons minne. Att ångra en ändring är en revert följd av en ny körning av verktyget.

Den andra är upprepbarhet. En välskriven definition är idempotent: tillämpad på en miljö som redan stämmer ändrar den ingenting, och tillämpad på en som har glidit iväg för den tillbaka miljön. Det finns två vägar dit: antingen justeras det som finns, eller så kastas det och ett nytt byggs upp från definitionen. Den andra vägen, gjord till en fast regel, är immutable infrastructure.

Den tredje är billiga miljöer. SP 800-204C beskriver verktygen som automatisering av infrastrukturens hela livslängd, från provisionering till avveckling, och påpekar att en del av den lever så kort att den inte är värd att underhålla. Ett team kan skapa flera testmiljöer vid behov och riva dem igen, och själva koden kan kontrolleras innan den rör något: vägledningen kräver att varje del av den skannas efter sårbarheter innan den mergas. En gren kan få en egen kortlivad kopia av produktionens uppsättning, och den som granskar kan se en förhandsvisning av ändringen innan den godkänns.

Kostnaderna förtjänar också att sägas rakt ut. Att ändra en brandväggsregel via en pull request går långsammare än att klicka i en webbkonsol, och under ett avbrott känns det dyrt. Varje verktyg har ett eget språk och en egen modell för tillstånd som måste läras in. Och en fil som kan bygga en miljö kan också radera den: spaken når lika långt åt båda hållen, och därför väger förhandsvisningen lika tungt som körningen efteråt.

Så ser det ut i praktiken

Här är en ändring på väg från redigering till körande system, i en uppsättning där ett CI-jobb kör verktyget och lägger torrkörningen på pull requesten:

```
Dag 1   09.10  Redigera nätverksdefinitionen: tillåt TCP 443 från kontoret
        09.12  Öppna en pull request; CI gör en torrkörning (planen)
        09.14  Planen syns på pull requesten: en regel tillagd, ingen borttagen
        10.30  En kollega läser planen såväl som diffen och godkänner
        10.31  Merge; CI genomför planen som blev godkänd
        10.33  En ny körning av samma definition rapporterar inga ändringar
Dag 40         Någon frågar varför 443 är öppen: commit-loggen säger vem
```

Planen är den del som gör sig förtjänt av sin plats. Diffen visar vad författaren ville ändra; planen visar vad verktyget faktiskt tänker göra, även sådant som författaren inte räknade med, till exempel en resurs som det tänker ta bort. Terraform kan spara en plan i en fil med `-out`, och `terraform apply` utför då operationerna i just den filen, så det som körs klockan 10.31 är det som kollegan godkände klockan 10.30. Den tysta körningen klockan 10.33 är kontrollen av att definitionen och systemet nu är överens.

Här på webbplatsen beskriver tjänstesidan om infrastruktur som kod upprepbar provisionering av servrar utifrån versionshanterad konfiguration i stället för att vara beroende av någons minne.

Att nå dit säkert är mest en fråga om ordning, och en portfoliosida från september 2026 visar varför. Den gäller en extra server hos en annan leverantör. Den tidigare maskinen hade satts upp för hand och skyddats senare, så under en tid var en nyskapad värd med bara en standardpolicy för lösenord öppen mot internet. Den här gången skapades servern från repot, och skyddet fanns på plats innan maskinen gjorde det. Brandväggens play körs innan serverns play, och hos den andra leverantören tilldelas taggarna före molnbrandväggen, eftersom regler riktade mot taggar inte har något att fästa vid förrän taggarna finns. En omkörning ställer inte till något, för om maskinen redan finns tar rollen den i bruk i stället för att skapa en till. Den rollen är den enda som kostar pengar, så dess egen dokumentation anger regionen, maskintypen med specifikation och priset per månad i euro. Sidan är också öppen med vad en torrkörning missar. Båda brandväggarna konfigureras genom anrop till leverantörernas REST-API:er från en generell HTTP-task, och den tasken stöder inte check mode. Därför hoppar en torrkörning över den. Den verkliga provkörningen är en körning av brandväggens play för sig, som inte kostar något och kan rullas tillbaka.

Var det går fel

Infrastrukturkod fallerar gärna på sätt som liknar framgång. Verktyget kör klart utan fel, CI-körningen är grön, och filerna och det verkliga systemet är i tysthet oense.

En snabb rättelse under en incident tas bort av nästa körning. Någon öppnar en port i konsolen klockan två på natten, avbrottet är över, och en vecka senare stänger den schemalagda körningen porten igen, eftersom filerna aldrig fick veta om den. Det omvända händer också: ingen kör verktyget igen, och filerna slutar beskriva verkligheten. Skriv in nödrättelsen i koden samma dag, och kör en schemalagd kontroll som jämför filerna med det som körs; SP 800-204C ber om den jämförelsen direkt efter varje driftsättning och med jämna mellanrum medan systemet är igång. Med Terraform gör plan-kommandots detaljerade slutkod det till ett kort nattligt jobb:

```
terraform plan -detailed-exitcode
# exit 0: inga ändringar, exit 1: fel, exit 2: ändringar finns
```

På en gren som redan har genomförts betyder slutkod 2 att det verkliga systemet har rört sig bort från koden, och att någon bör ta reda på varför.

Att ta bort en resurs ur filen tar inte alltid bort den. Verktygen är oense om vad en saknad resurs betyder. För Azure Resource Manager-mallar skriver Microsoft i dokumentationen för what-if att en resurs som finns men inte står i mallen bara raderas i complete mode. I andra lägen fortsätter den att köra och kosta pengar. Lär dig verktygets regel för borttagning innan du litar på den, och läs förhandsvisningen efter de raderingar du väntar dig såväl som dem du fruktar.

Torrkörningen ser inte allt. En förhandsvisning är en modell av ändringen, ingen garanti. Microsofts what-if-dokumentation nämner egenskaper som rapporteras som raderade fast Azure bara fyller i ett standardvärde under driftsättningen, och kallar resultatet brus; varje egenskap som sätts via funktionen `reference` visas som ändrad vid varje körning. Portfoliosidan ovan visar det motsatta hålet: en task utan stöd för check mode hoppas över i en torrkörning, så förhandsvisningen säger ingenting om den. Känn till verktygets kända brus, och räkna allt som förhandsvisningen inte ser som otestat tills en provkörning mot något billigt som går att ångra har täckt det.

Hemligheter hamnar i repot eller i tillståndet. Ett lösenord som skrivs in i en definitionsfil blir kvar i historiken i varenda klon. Mindre uppenbart: Terraforms dokumentation varnar för att tillståndet kan innehålla hemligheter, och som standard är det tillståndet en lokal fil med namnet `terraform.tfstate`. Håll hemligheter i ett hemlighetslager och hänvisa till dem med namn, lägg tillståndet i fjärrlagring med låsning och åtkomstkontroll, som HashiCorp rekommenderar, och redigera aldrig tillståndsfilen för hand; kommandot `terraform state` finns för de ändringar som behövs.

Ett skript utger sig för att vara en definition. Ett skalskript i repot som skapar en server står under versionshantering, men det är inte idempotent: kör det två gånger, så finns det två servrar, eller ett fel mitt i alltihop. Använd deklarativa resurser eller steg som kontrollerar innan de agerar, som rollen ovan som tar över en befintlig maskin. Ansibles dokumentation sätter ribban tydligt: när ett system redan är i det tillstånd som en playbook beskriver, ändrar en ny körning av playbooken ingenting.

Läs mer

- Implementation of DevSecOps for a Microservices-based Application with Service Mesh (NIST SP 800-204C) — deklarativ, versionshanterad infrastrukturkod, avvikelser och skanning.
- Terraform Language Documentation — det deklarativa språket, dess resurser och dess block.
- Command: plan — torrkörningen, sparade planer och de detaljerade slutkoderna.
- State — vad Terraform registrerar, var det ligger och varför det är känsligt.
- Introduction to Ansible — agentlös automatisering över SSH, playbooks och önskat tillstånd.
- Template deployment what-if — förhandsvisning av en driftsättning i Azure, ändringstyperna och det kända bruset.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är infrastruktur som kod?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

welcome@engineer.company +45 50 10 61 84 Köpenhamn, Danmark

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/infrastructure-as-code/