**Engineer**

# Terraform

HashiCorps verktøy som planlegger og ruller ut infrastruktur

## Hva det er

Terraform er HashiCorps verktøy for infrastruktur som kode. Du beskriver ressurser i skyen og i eget datasenter i vanlige tekstfiler, holder filene under versjonskontroll, og Terraform regner selv ut hvilke ekte objekter som må opprettes, endres eller slettes for at det som kjører, skal stemme med filene. Selve programmet kalles som `terraform`, med små bokstaver, og det er slik navnet står i hver kommando og hvert skript.

Språket er deklarativt. HashiCorps dokumentasjon beskriver det som et språk som angir et mål, og ikke stegene som fører dit. En konfigurasjon sier at det finnes en maskin av en viss størrelse i et visst nettverk; den lister ikke opp API-kallene som skal til for å lage den. Filene har endelsen `.tf`, og det finnes også en JSON-basert variant med endelsen `.tf.json`. Den opprinnelige syntaksen er definert ut fra en syntaks som heter HCL, og har to hovedbyggesteiner: en blokk, som er en beholder for annet innhold, og et argument, som gir et navn en verdi.

Tre ideer bærer resten av verktøyet.

Providere er programtillegg. Konfigurasjonen forteller Terraform hvilke programtillegg som skal installeres, og `terraform init` finner, laster ned og installerer dem. HashiCorp skriver at providere lar Terraform arbeide med praktisk talt enhver plattform eller tjeneste som har et tilgjengelig API, og de offentlig tilgjengelige står i Terraform Registry. En provider definerer ressurstypene for sin plattform, slik at det samme kjerneprogrammet kan styre en DNS-sone, en virtuell maskin og en overvåkingsregel hos tre ulike leverandører.

Moduler er kataloger. En modul er en samling `.tf` - og `.tf.json` -filer som ligger sammen i én katalog, og rotmodulen er arbeidskatalogen der Terraform startes.

Tilstand er hukommelse. Terraform holder rede på den ekte infrastrukturen i en tilstandsfil, som innføringen i dokumentasjonen kaller sannhetskilden for miljøet. Filen kobler hver ressurs i konfigurasjonen til det ekte objektet den styrer, sammen med metadata. Som standard er den en lokal fil som heter `terraform.tfstate` , og den forrige versjonen ligger ved siden av som `terraform.tfstate.backup` .

Lisensen er endret. Den 10. august 2023 kunngjorde HashiCorp at kildekoden gikk over fra Mozilla Public License 2.0 til Business Source License 1.1 for alle framtidige utgivelser av produktene, mens API-er, SDK-er og nesten alle andre biblioteker blir værende på MPL 2.0. OpenTofu omtaler seg selv som en fork av Terraform, startet av Gruntwork, Spacelift, Harness, Env0, Scalr og andre, og utviklet under forvaltning av Linux Foundation. Prosjektets repo er publisert under MPL 2.0, og nettstedet presenterer OpenTofu som en erstatning for Terraform som beholder eksisterende arbeidsflyter og konfigurasjoner.

Bruk begrepet i en beslutning

Bruk «Hva er Terraform?» 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

HashiCorp beskriver kjernearbeidsflyten i tre steg. Write: du definerer ressurser, som kan spenne over flere providere og tjenester. Plan: Terraform lager en kjøreplan som beskriver hva det vil opprette, oppdatere eller slette. Apply: etter godkjenning utfører Terraform de planlagte operasjonene i riktig rekkefølge og tar hensyn til avhengighetene mellom ressursene, som det finner ved å bygge en ressursgraf.

Det meste av verdien ligger i det midterste steget. En plan er en forhåndsvisning som en person som ikke skrev endringen, kan lese før noe skjer. Den som går gjennom endringen, ser mer enn forskjellen mellom filene, nemlig konsekvensene: to ressurser opprettet, én oppdatert og av og til én slettet som ingen hadde tenkt å røre. Den siste linjen er billig å oppdage i en pull request og svært dyr å oppdage etterpå.

Den andre grunnen er rekkevidde. Fordi kunnskapen om hver plattform ligger i dens provider, dekker ett språk, én arbeidsflyt og ett sett gjennomgangsvaner alle providerne et team bruker. Et selskap med to skyer og en egen DNS-leverandør kan beskrive alle tre i ett repo og gå gjennom alle tre på samme måte. Bicep fra Microsoft gjør en annen avveining i arbeidsflyten med Azure Resource Manager: det fører ingen egen tilstandsfil, fordi Azure lagrer utrullingstilstanden. Det gjør ikke Bicep til et språk som bare dekker Azure: utvidelsene støtter også ressurser utenfor Resource Manager, blant annet Kubernetes og Microsoft Graph.

Den tredje grunnen er tilstanden. Siden Terraform registrerer hvilket ekte objekt som hører til hvilken blokk, kan det skille et objekt det styrer fra et det aldri har sett, og det kan sammenligne sine opptegnelser med det som kjører. De opptegnelsene er samtidig prisen for verktøyet. De må ligge et sted hele teamet når, de trenger en lås så to kjøringer ikke skriver samtidig, og de kan inneholde hemmeligheter, så tilgangen til dem må styres like strengt som til hemmelighetene selv.

Det finnes mindre kostnader også. HCL er ett språk til å lære, og hver provider har sine egne ressurstyper og særheter. Team som bygger et produkt oppå Terraform, har også en lisens å lese: HashiCorps lisens-FAQ har et eget spørsmål om hva som regnes som et konkurrerende tilbud. OpenTofu tilbyr den samme arbeidsflyten under MPL 2.0.

## Slik ser det ut i praksis

HashiCorps side om ressurssyntaks bruker denne blokken som eksempel:

```
resource "aws_instance" "web" {
  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"
}
```

`resource` er blokktypen, og `aws_instance` er ressurstypen, som AWS-provideren definerer. `"web"` er det lokale navnet: Terraform bruker den etiketten til å følge ressursen i tilstandsfilen, og den har ingen virkning på innstillingene til den ekte maskinen. De to linjene inne i blokken er argumenter.

Tre kommandoer tar filen fra tekst til en kjørende maskin:

```
terraform init
terraform plan -out=tfplan
terraform apply tfplan
```

`terraform init` gjør arbeidskatalogen klar: den installerer provider-programtilleggene som konfigurasjonen ber om, og setter opp stedet der tilstanden skal ligge. Den er trygg å kjøre flere ganger, og dokumentasjonen sier at låsefilen for avhengigheter skal sjekkes inn i versjonskontroll.

`terraform plan -out=tfplan` beregner planen og skriver den til en fil i et ugjennomsiktig format. Planleggingen utfører ikke de foreslåtte infrastrukturendringene. Deretter utfører `terraform apply tfplan` operasjonene i den filen uten å be om bekreftelse, fordi HashiCorp regner det å sende inn en lagret plan som godkjenningen. Av samme grunn blir flagget `-auto-approve` ignorert når en planfil er oppgitt.

I et team faller delingen naturlig. En CI-jobb kjører planen på pull requesten og tar vare på filen, en person leser planen i tillegg til forskjellen i filene, og etter sammenslåingen bruker en annen jobb nettopp den filen, slik at det som kjøres, er det som ble lest. Dokumentasjonen for apply legger til en betingelse som er lett å glemme: sørg for at ingen kan endre infrastrukturen utenom Terraform-arbeidsflyten, ellers beskriver planen en verden som allerede har flyttet seg.

Her på nettstedet setter teknologisiden Terraform ved siden av Ansible: begge skriver ned hvordan maskinene skal være, slik at et miljø kan leses i stedet for å måtte rekonstrueres. Et oppdrag i porteføljen bruker ideen med et annet verktøy: Azure-infrastrukturen ble satt opp som kode i Bicep. Miljøene testing, staging og product bygges alle fra ett sett maler, og siden malene tar parametere, er et nytt eller endret miljø en pull request. Ved siden av denne artikkelen viser oppdraget avveiningen fra tidligere. Bicep har ingen egen tilstandsfil fordi Azure selv holder oversikten, mens Terraform fører sin egen tilstand og til gjengjeld kan brukes på tvers av flere skyer.

# Hvor det går galt

De fleste Terraform-uhell har samme form: konfigurasjonen er gyldig, kjøringen lykkes, og noe ekte er borte eller endret.

**En omdøpt ressurs blir slettet.** Endre etiketten `"web"` til `"frontend"`, og ressursen får en ny adresse. HashiCorps veiledning om refaktorering sier at Terraform som standard tolker en endret adresse som en instruks om å slette det eksisterende objektet og opprette et nytt. For en maskin som kan kastes, koster det noen minutter; for en database koster det dataene. Løsningen er en `moved`-blokk, som oppdaterer adressen uten å slette noe:

```
moved {
  from = aws_instance.web
  to   = aws_instance.frontend
}
```

Behold blokken når den har gjort jobben sin. HashiCorp anbefaler sterkt å beholde gamle `moved`-blokker og skriver at det å fjerne en slik blokk er en bakoverinkompatibel endring.

**To personer kjører apply mot hver sin lokale tilstand.** I utgangspunktet er tilstanden til hvert workspace en lokal fil. Det passer for én person på én bærbar maskin. Så snart en kollega kjører Terraform fra en annen kopi, er de to tilstandsfilene uenige om hva som finnes, og hver kjøring handler ut fra sin egen versjon. Løsningen er å holde tilstanden i HCP Terraform eller i fjernlagring med låsing og tilgangskontroll, slik HashiCorp anbefaler. Terraform låser da tilstanden automatisk for hver operasjon som kan skrive til den, og fortsetter ikke hvis låsen ikke kan tas. Ikke all lagring støtter låsing, så sjekk det før du velger, og la `-lock=false` være: flagget finnes, men dokumentasjonen fraråder det.

**En hengende lås brytes mens noen fortsatt arbeider.** Når den automatiske opplåsingen svikter, blir låsen liggende, og kommandoen `force-unlock` frigjør den med lås-ID-en som Terraform skrev ut. Dokumentasjonen om låsing ber brukeren være svært forsiktig: å bryte en lås som noen andre holder, kan gi to skrivere mot samme tilstand, og kommandoen er bare ment for din egen lås etter at den automatiske opplåsingen har sviktet. Løsningen er å finne ut hvem låsen tilhører og om den kjøringen fortsatt lever, før noe tvinges.

**Tilstandsfilen lekker hemmeligheter.** HashiCorp advarer om at hemmeligheter kan havne i tilstanden, og at de kan bli eksponert hvis tilstanden lagres i et system uten sikker tilgangskontroll. En tilstandsfil i en bøtte alle kan lese, eller kopiert inn i en byggeartefakt, er en lekkasje av påloggingsdata. Hold den utenfor repoet, gi den samme tilgangskontroll som hemmelighetene den inneholder, og rediger den aldri for hånd; kommandoen `terraform state` finnes for de rettelsene som trengs.

**Godkjenningen blir en formalitet.** Flagget `-auto-approve` hopper over den interaktive godkjenningen av planen, og en lagret planfil hopper også over den, fordi filen regnes som godkjenningen. Dokumentasjonen for apply påpeker at dette også gjelder destruktive operasjoner som sletting av ressurser. En CI-jobb som kjører apply etter hver sammenslåing uten en gjennomgått plan, har fjernet det ene øyeblikket der en person kunne ha sett slettingen komme. Løsningen er flyten med lagret plan som er beskrevet ovenfor, med påkrevd gjennomgang av et menneske mellom plan-jobben og apply-jobben. Der `-auto-approve` brukes i automatisering, er HashiCorps eget råd å kjøre en plan først og gå gjennom den.

## Les mer

- What is Terraform? — HashiCorps innføring: write, plan, apply, providere og tilstand.
- terraform apply command — lagrede planer, `-auto-approve` og kommandoen i automatisering.
- State Locking — automatiske låser, lagring uten låsing og advarselen om `force-unlock`.
- Refactor modules — hvorfor en endret adresse betyr sletting og nyopprettelse, og `moved` -blokken.
- HashiCorp adopts Business Source License — kunngjøringen fra august 2023 om overgangen fra MPL 2.0 til BSL 1.1.
- OpenTofu — forken av Terraform som utvikles under forvaltning av Linux Foundation.

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 Terraform?» 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/terraform/