Engineer

# Terraform

HashiCorps værktøj, der planlægger og udruller infrastruktur

## Hvad det er

Terraform er HashiCorps værktøj til infrastruktur som kode. Man beskriver ressourcer i skyen og i eget datacenter i almindelige tekstfiler, holder filerne i versionsstyring, og Terraform regner ud, hvilke rigtige objekter der skal oprettes, ændres eller slettes, for at det kørende miljø svarer til filerne. Selve programmet kaldes som `terraform` med små bogstaver, og sådan står navnet i alle kommandoer og scripts.

Sproget er deklarativt. HashiCorps dokumentation beskriver det som et sprog, der angiver et mål og ikke de trin, der fører derhen. En konfiguration siger, at der findes en maskine af en bestemt størrelse i et bestemt netværk; den opremser ikke de API-kald, der skulle til for at skabe den. Filerne har endelsen `.tf`, og der findes også en JSON-baseret variant med endelsen `.tf.json`. Den oprindelige syntaks er defineret ud fra en syntaks ved navn HCL og har to grundelementer: en blok, som er en beholder for andet indhold, og et argument, som giver et navn en værdi.

Tre idéer bærer resten af værktøjet.

Providers er plugins. Konfigurationen fortæller Terraform, hvilke plugins der skal installeres, og `terraform init` finder, henter og installerer dem. HashiCorp skriver, at providers lader Terraform arbejde med stort set enhver platform eller tjeneste, der har et tilgængeligt API, og de offentligt tilgængelige står i Terraform Registry. En provider definerer ressourcetyperne for sin platform, så det samme kerneprogram kan styre en DNS-zone, en virtuel maskine og en overvågningsregel hos tre forskellige leverandører.

Moduler er mapper. Et modul er en samling `.tf` - og `.tf.json` -filer, der ligger sammen i én mappe, og rodmodulet er den arbejdsmappe, Terraform startes fra.

Tilstanden er hukommelsen. Terraform holder styr på den rigtige infrastruktur i en tilstandsfil, som introduktionen kalder sandhedskilden for miljøet. Filen kobler hver ressource i konfigurationen til det rigtige objekt, den styrer, og gemmer metadata ved siden af. Som standard er det en lokal fil ved navn `terraform.tfstate`, og den forrige udgave ligger ved siden af som `terraform.tfstate.backup`.

Licensen har skiftet. Den 10. august 2023 meddelte HashiCorp, at selskabet flyttede sin kildekode fra Mozilla Public License 2.0 til Business Source License 1.1 for alle fremtidige udgivelser af sine produkter, mens API'er, SDK'er og næsten alle andre biblioteker forbliver under MPL 2.0. OpenTofu beskriver sig selv som en fork af Terraform, startet af Gruntwork, Spacelift, Harness, Env0, Scalr og andre og udviklet under Linux Foundations ansvar. Projektets repository er udgivet under MPL 2.0, og hjemmesiden præsenterer det som en erstatning for Terraform, der bevarer eksisterende arbejdsgange og konfigurationer.

Brug begrebet i en beslutning

Brug »Hvad er Terraform?« 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

HashiCorp beskriver kernearbejdsgangen i tre trin. Write: man definerer ressourcerne, som kan spænde over flere providers og tjenester. Plan: Terraform laver en udførelsesplan, der beskriver, hvad det vil oprette, opdatere eller slette. Apply: når planen er godkendt, udfører Terraform handlingerne i den rigtige rækkefølge og tager hensyn til afhængighederne mellem ressourcerne, som det finder ved at bygge en ressourcegraf.

Det midterste trin er der, hvor det meste af værdien ligger. En plan er en forhåndsvisning, som en person, der ikke har skrevet ændringen, kan læse, før noget sker. En reviewer ser mere end forskellen mellem filerne: konsekvenserne, to ressourcer oprettet, én opdateret og en gang imellem én slettet, som ingen havde tænkt sig at røre. Den sidste linje er billig at fange i en pull request og meget dyr at opdage bagefter.

Den anden grund er rækkevidden. Fordi viden om hver platform ligger i dens provider, dækker ét sprog, én arbejdsgang og én måde at reviewe på alle de providers, et team bruger. En virksomhed med to skyer og en separat DNS-udbyder kan beskrive alle tre i ét repository og gennemgå dem på samme måde. Microsofts Bicep foretager en anden afvejning i sin arbejdsgang med Azure Resource Manager: det fører ingen egen tilstandsfil, fordi Azure gemmer udrulningens tilstand. Det gør ikke Bicep til et sprog, der kun dækker Azure: dets udvidelser understøtter også ressourcer uden for Resource Manager, blandt andet Kubernetes og Microsoft Graph.

Den tredje grund er tilstanden. Fordi Terraform registrerer, hvilket rigtigt objekt der hører til hvilken blok, kan det skelne et objekt, det styrer, fra et, det aldrig har set, og det kan sammenligne optegnelsen med det, der kører. Den optegnelse er også prisen for værktøjet. Den skal ligge et sted, hele teamet kan nå, den skal have en lås, så to kørsler ikke skriver i den samtidig, og den kan indeholde hemmeligheder, så den kræver lige så stram adgangskontrol som dem.

Dertil kommer mindre omkostninger. HCL er endnu et sprog at lære, og hver provider har sine egne ressourcetyper og særheder. Teams, der bygger et produkt oven på Terraform, har også en licens at læse: HashiCorps licens-FAQ har et spørgsmål for sig om, hvad der tæller som et konkurrerende tilbud. OpenTofu tilbyder den samme arbejdsgang under MPL 2.0.

## Sådan ser det ud i praksis

HashiCorps side om ressourcesyntaks bruger denne blok som eksempel:

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

`resource` er bloktypen, og `aws_instance` er ressourcetypen, som AWS-provideren definerer. `"web"` er det lokale navn: Terraform bruger den etiket til at følge ressourcen i tilstandsfilen, og den har ingen indflydelse på den rigtige maskines indstillinger. De to linjer inde i blokken er argumenter.

Tre kommandoer fører filen fra tekst til en kørende maskine:

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

`terraform init` gør arbejdsmappen klar: den installerer de provider-plugins, konfigurationen beder om, og sætter det sted op, hvor tilstanden skal ligge. Den kan trygt køres flere gange, og dokumentationen siger, at afhængighedslåsefilen skal lægges i versionsstyring.

`terraform plan -out=tfplan` beregner planen og skriver den til en fil i et uigennemsigtigt format. Planlægningen anvender ikke de foreslåede infrastrukturændringer. `terraform apply tfplan` udfører derefter handlingerne i filen uden at bede om bekræftelse, fordi HashiCorp regner det at give en gemt plan videre som selve godkendelsen. Af samme grund ignoreres flaget `-auto-approve`, når der gives en planfil.

I et team falder opdelingen af sig selv. Et CI-job kører planen på pull requesten og gemmer filen, en person læser planen og ikke kun forskellen i filerne, og efter merge anvender et andet job præcis den fil, så det, der kører, er det, der blev læst. Dokumentationen for apply tilføjer en betingelse, der er let at glemme: sørg for, at ingen kan ændre infrastrukturen uden om Terraforms arbejdsgang, ellers beskriver planen en verden, der allerede har flyttet sig.

På dette site sætter siden om teknologier Terraform ved siden af Ansible: begge skriver ned, hvordan maskinerne skal se ud, så et miljø kan læses i stedet for at skulle rekonstrueres. Et portfolio-eksempel bruger idéen med et andet værktøj: Azure-infrastruktur blev provisioneret som kode i Bicep. Miljøerne testing, staging og product bygges alle af ét sæt skabeloner, og da skabelonerne tager parametre, er et nyt eller ændret miljø en pull request. Ved siden af denne artikel viser eksemplet afvejningen fra før. Bicep har ingen tilstandsfil, fordi Azure selv holder styr på ressourcerne, mens Terraform fører sin egen tilstand og til gengæld kan bruges på tværs af flere skyer.

## Hvor det går galt

De fleste uheld med Terraform har samme form: konfigurationen er gyldig, kørslen lykkes, og noget rigtigt er forsvundet eller ændret.

**En omdøbt ressource bliver slettet.** Skift etiketten `"web"` til `"frontend"`, og ressourcen har en ny adresse. HashiCorps vejledning om refaktorering siger, at Terraform som standard opfatter en ændret adresse som en instruks om at slette det eksisterende objekt og oprette et nyt. For en maskine, man kan smide væk, koster det et par minutter; for en database koster det dataene. Løsningen er en `moved`-blok, som opdaterer adressen uden at slette noget:

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

Lad blokken blive stående, når den har gjort sit arbejde. HashiCorp anbefaler kraftigt at beholde gamle `moved`-blokke og skriver, at det er en bagudinkompatibel ændring at fjerne én.

**To personer kører apply mod hver sin lokale tilstand.** Fra start er hvert workspaces tilstand en lokal fil. Det passer til én person på én bærbar. Så snart en kollega kører Terraform fra en anden kopi, er de to tilstandsfiler uenige om, hvad der findes, og hver kørsel handler ud fra sin egen udgave. Løsningen er at lægge tilstanden i HCP Terraform eller i fjernlager med låsning og adgangskontrol, som HashiCorp anbefaler. Terraform låser så automatisk tilstanden ved hver handling, der kan skrive i den, og fortsætter ikke, hvis det ikke kan få låsen. Ikke alle lagre understøtter låsning, så tjek det, før du vælger et, og lad `-lock=false` være: flaget findes, og dokumentationen fraråder det.

**En hængende lås bliver brudt, mens en anden stadig arbejder.** Når den automatiske oplåsning slår fejl, bliver låsen hængende, og kommandoen `force-unlock` frigiver den med det lås-id, Terraform skrev ud. Dokumentationen om låsning beder brugeren være meget forsigtig: bryder man en lås, som en anden holder, kan der ende med at være to, der skriver i den samme tilstand, og kommandoen er kun beregnet til ens egen lås, når den automatiske oplåsning er slået fejl. Løsningen er at finde ud af, hvis lås det er, og om den kørsel stadig er i live, før man tvinger noget.

**Tilstandsfilen lækker hemmeligheder.** HashiCorp advarer om, at hemmeligheder kan havne i tilstanden, og at de kan blive eksponeret, hvis den ligger i et system uden sikker adgangskontrol. En tilstandsfil i en bucket, alle kan læse, eller kopieret ind i et build-artefakt, er en lækage af adgangsoplysninger.

Hold den ude af repositoriet, giv den samme adgangskontrol som de hemmeligheder, den indeholder, og redigér den aldrig i hånden; kommandoen `terraform state` findes til de rettelser, der er brug for.

**Godkendelsen bliver en formalitet.** Flaget `-auto-approve` springer den interaktive godkendelse af planen over, og en gemt planfil gør det samme, fordi filen tæller som godkendelsen. Dokumentationen for apply gør opmærksom på, at det også gælder destruktive handlinger som sletning af ressourcer. Et CI-job, der kører apply ved hver merge uden en gennemgået plan, har fjernet det ene øjeblik, hvor en person kunne have set sletningen komme. Løsningen er arbejdsgangen med den gemte plan ovenfor med et påkrævet menneskeligt review mellem plan-jobbet og apply-jobbet. Hvor `-auto-approve` bruges i automatisering, er HashiCorps eget råd at køre en plan først og gennemgå den.

## Læs mere

- What is Terraform? — HashiCorps introduktion: write, plan, apply, providers og tilstand.
- terraform apply command — gemte planer, `-auto-approve` og kommandoen i automatisering.
- State Locking — automatiske låse, lagre uden låsning og advarslen om `force-unlock`.
- Refactor modules — hvorfor en ændret adresse betyder slet og opret, og `moved`-blokken.
- HashiCorp adopts Business Source License — meddelelsen fra august 2023 om skiftet fra MPL 2.0 til BSL 1.1.
- OpenTofu — forken af Terraform, der vedligeholdes under Linux Foundation.

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 Terraform?« 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/terraform/