Engineer

Terraform

HashiCorps verktyg som planerar och rullar ut infrastruktur

Vad det är

Terraform är HashiCorps verktyg för infrastruktur som kod. Du beskriver resurser i molnet och i det egna datacentret i vanliga textfiler, håller filerna under versionshantering, och Terraform räknar själv ut vilka riktiga objekt som ska skapas, ändras eller raderas för att det som körs ska stämma med filerna. Själva programmet anropas som `terraform`, med gemener, och det är så namnet står i varje kommando och varje skript.

Språket är deklarativt. HashiCorps dokumentation beskriver det som ett språk som anger ett mål och inte stegen dit. En konfiguration säger att det finns en maskin av en viss storlek i ett visst nätverk; den räknar inte upp de API-anrop som behövs för att skapa den. Filerna har ändelsen `.tf`, och det finns också en JSON-baserad variant med ändelsen `.tf.json`. Den ursprungliga syntaxen är definierad utifrån en syntax som heter HCL och har två huvudsakliga byggstenar: ett block, som är en behållare för annat innehåll, och ett argument, som ger ett namn ett värde.

Tre idéer bär resten av verktyget.

Providers är insticksprogram. Konfigurationen talar om för Terraform vilka insticksprogram som ska installeras, och `terraform init` hittar, laddar ner och installerar dem. HashiCorp skriver att providers låter Terraform arbeta med praktiskt taget varje plattform eller tjänst som har ett åtkomligt API, och de offentligt tillgängliga finns i Terraform Registry. En provider definierar resurstyperna för sin plattform, så att samma kärnprogram kan hantera en DNS-zon, en virtuell maskin och en övervakningsregel hos tre olika leverantörer.

Moduler är kataloger. En modul är en samling `.tf` - och `.tf.json` -filer som ligger tillsammans i en katalog, och rotmodulen är arbetskatalogen där Terraform startas.

Tillstånd är minne. Terraform håller reda på den riktiga infrastrukturen i en tillståndsfil, som introduktionen i dokumentationen kallar sanningskällan för miljön. Filen kopplar varje resurs i konfigurationen till det riktiga objekt den hanterar, tillsammans med metadata. Som standard är den en lokal fil med namnet `terraform.tfstate`, och den förra versionen ligger bredvid som `terraform.tfstate.backup`.

Licensen har ändrats. Den 10 augusti 2023 meddelade HashiCorp att källkoden gick över från Mozilla Public License 2.0 till Business Source License 1.1 för alla framtida versioner av produkterna, medan API:er, SDK:er och nästan alla andra bibliotek blir kvar under MPL 2.0. OpenTofu beskriver sig som en fork av Terraform, startad av Gruntwork, Spacelift, Harness, Env0, Scalr med flera, och utvecklad under förvaltning av Linux Foundation. Projektets repo är publicerat under MPL 2.0, och webbplatsen presenterar OpenTofu som en ersättning för Terraform som behåller befintliga arbetsflöden och konfigurationer.

Använd begreppet i ett beslut

Använd ”Vad är Terraform?” 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

HashiCorp beskriver kärnarbetsflödet i tre steg. Write: du definierar resurser, som kan sträcka sig över flera providers och tjänster. Plan: Terraform tar fram en körplan som beskriver vad det kommer att skapa, uppdatera eller radera. Apply: efter godkännande utför Terraform de planerade operationerna i rätt ordning och tar hänsyn till beroendena mellan resurserna, som det räknar ut genom att bygga en resursgraf.

Det mesta av värdet ligger i det mellersta steget. En plan är en förhandsvisning som någon som inte skrev ändringen kan läsa innan något händer. Den som granskar ser mer än skillnaden mellan filerna, nämligen följderna: två resurser skapade, en uppdaterad och då och då en raderad som ingen hade tänkt röra. Den sista raden är billig att upptäcka i en pull request och mycket dyr att upptäcka efteråt.

Det andra skälet är räckvidd. Eftersom kunskapen om varje plattform ligger i dess provider täcker ett språk, ett arbetsflöde och en uppsättning granskningsvanor alla providers som ett team använder. Ett företag med två moln och en separat DNS-leverantör kan beskriva alla tre i ett repo och granska dem på samma sätt. Microsofts Bicep gör en annan avvägning i arbetsflödet med Azure Resource Manager: det för ingen egen tillståndsfil, eftersom Azure lagrar driftsättningens tillstånd. Det gör inte Bicep till ett språk som bara täcker Azure: dess tillägg stöder även resurser utanför Resource Manager, bland annat Kubernetes och Microsoft Graph.

Det tredje skälet är tillståndet. Eftersom Terraform registrerar vilket riktigt objekt som hör till vilket block kan det skilja ett objekt det hanterar från ett som det aldrig har sett, och det kan jämföra sina anteckningar med det som körs. De anteckningarna är samtidigt priset för verktyget. De måste ligga där hela teamet når dem, de behöver ett lås så att två körningar inte skriver samtidigt, och de kan innehålla hemligheter, så åtkomsten till dem måste styras lika strikt som till hemligheterna själva.

Det finns mindre kostnader också. HCL är ytterligare ett språk att lära sig, och varje provider har sina egna resurstyper och egenheter. Team som bygger en produkt ovanpå Terraform har dessutom en licens att läsa: HashiCorps licens-FAQ har en egen fråga om vad som räknas som ett konkurrerande erbjudande. OpenTofu erbjuder samma arbetsflöde under MPL 2.0.

Så ser det ut i praktiken

HashiCorps sida om resurssyntax använder det här blocket som exempel:

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

`resource` är blocktypen och `aws_instance` är resurstypen, som AWS-providern definierar. `"web"` är det lokala namnet: Terraform använder den etiketten för att följa resursen i tillståndsfilen, och den påverkar inte inställningarna för den riktiga maskinen. De två raderna inuti blocket är argument.

Tre kommandon tar filen från text till en körande maskin:

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

`terraform init` förbereder arbetskatalogen: det installerar de provider-insticksprogram som konfigurationen ber om och ställer i ordning platsen där tillståndet ska ligga. Kommandot är säkert att köra flera gånger, och dokumentationen säger att låsfilen för beroenden ska checkas in i versionshanteringen.

`terraform plan -out=tfplan` beräknar planen och skriver den till en fil i ett ogenomskinligt format. Planeringen genomför inte de föreslagna infrastrukturändringarna. Därefter utför `terraform apply tfplan` operationerna i den filen utan att be om bekräftelse, eftersom HashiCorp räknar överlämnandet av en sparad plan som godkännandet. Av samma skäl ignoreras flaggan `-auto-approve` när en planfil anges.

I ett team faller uppdelningen sig naturligt. Ett CI-jobb kör planen på pull requesten och sparar filen, en person läser planen och inte bara skillnaden i filerna, och efter sammanslagningen tillämpar ett andra jobb just den filen, så att det som körs är det som lästes. Dokumentationen för apply lägger till ett villkor som är lätt att glömma: se till att ingen kan ändra infrastrukturen utanför Terraforms arbetsflöde, annars beskriver planen en värld som redan har flyttat sig.

Här på webbplatsen ställer sidan om teknikstacken Terraform bredvid Ansible: båda skriver ner hur maskinerna ska vara, så att en miljö kan läsas i stället för att behöva rekonstrueras. Ett uppdrag i portfolion använder idén med ett annat verktyg: Azure-infrastrukturen sattes upp som kod i Bicep. Miljöerna testing, staging och product byggs alla från en och samma uppsättning mallar, och eftersom mallarna tar parametrar är en ny eller ändrad miljö en pull request.

Bredvid den här artikeln visar uppdraget avvägningen från tidigare. Bicep har ingen egen tillståndsfil eftersom Azure själv håller reda på resurserna, medan Terraform för sitt eget tillstånd och i gengäld kan användas med flera moln.

Var det går fel

De flesta Terraform-olyckor har samma form: konfigurationen är giltig, körningen lyckas, och något riktigt är borta eller ändrat.

En resurs som byter namn raderas. Ändra etiketten `"web"` till `"frontend"`, så får resursen en ny adress. HashiCorps vägledning om refaktorering säger att Terraform som standard tolkar en ändrad adress som en instruktion att radera det befintliga objektet och skapa ett nytt. För en maskin som går att slänga kostar det några minuter; för en databas kostar det datan. Lösningen är ett `moved`-block, som uppdaterar adressen utan att radera något:

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

Behåll blocket när det har gjort sitt. HashiCorp rekommenderar starkt att gamla `moved`-block behålls och skriver att det är en bakåtinkompatibel ändring att ta bort ett sådant block.

Två personer kör apply mot var sitt lokala tillstånd. Från början är tillståndet för varje workspace en lokal fil. Det passar en person på en bärbar dator. Så snart en kollega kör Terraform från en annan kopia är de två tillståndsfilerna oense om vad som finns, och varje körning agerar utifrån sin egen version. Lösningen är att hålla tillståndet i HCP Terraform eller i fjärrlagring med låsning och åtkomstkontroll, som HashiCorp rekommenderar. Terraform låser då tillståndet automatiskt för varje operation som kan skriva till det och fortsätter inte om låset inte går att ta. All lagring stöder inte låsning, så kontrollera det innan du väljer, och låt `-lock=false` vara: flaggan finns, men dokumentationen avråder från den.

Ett hängande lås bryts medan någon fortfarande arbetar. När den automatiska upplåsningen misslyckas blir låset kvar, och kommandot `force-unlock` släpper det med det lås-ID som Terraform skrev ut. Dokumentationen om låsning ber

användaren att vara mycket försiktig: att bryta ett lås som någon annan håller kan ge två skrivare mot samma tillstånd, och kommandot är bara avsett för ditt eget lås efter att den automatiska upplåsningen har misslyckats. Lösningen är att ta reda på vems låset är och om den körningen fortfarande lever innan något tvingas upp.

Tillståndsfilen läcker hemligheter. HashiCorp varnar för att hemligheter kan hamna i tillståndet och att de kan exponeras om tillståndet lagras i ett system utan säker åtkomstkontroll. En tillståndsfil i en bucket som alla kan läsa, eller kopierad in i en byggartefakt, är en läcka av inloggningsuppgifter. Håll den utanför repot, ge den samma åtkomstkontroll som hemligheterna den innehåller, och redigera den aldrig för hand; kommandot `terraform state` finns för de rättelser som behövs.

Godkännandet blir en formalitet. Flaggan `-auto-approve` hoppar över det interaktiva godkännandet av planen, och en sparad planfil hoppar också över det, eftersom filen räknas som godkännandet. Dokumentationen för apply påpekar att detta även gäller destruktiva operationer som att radera resurser. Ett CI-jobb som kör apply efter varje sammanslagning utan en granskad plan har tagit bort det enda ögonblick då en person kunde ha sett raderingen komma. Lösningen är flödet med sparad plan som beskrevs ovan, med obligatorisk granskning av en människa mellan plan-jobbet och apply-jobbet. Där `-auto-approve` används i automatisering är HashiCorps eget råd att först köra en plan och granska den.

Läs mer

- What is Terraform? — HashiCorps introduktion: write, plan, apply, providers och tillstånd.
- terraform apply command — sparade planer, `-auto-approve` och kommandot i automatisering.
- State Locking — automatiska lås, lagring utan låsning och varningen om `force-unlock`.
- Refactor modules — varför en ändrad adress betyder radering och nyskapande, och `moved`-blocket.
- HashiCorp adopts Business Source License — beskedet från augusti 2023 om bytet från MPL 2.0 till BSL 1.1.
- OpenTofu — forken av Terraform som utvecklas under förvaltning av Linux Foundation.

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 Terraform?” 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/terraform/