Engineer

Terraform

HashiCorps Werkzeug, das Infrastrukturcode plant und umsetzt

Was es ist

Terraform ist HashiCorps Werkzeug für Infrastruktur als Code. Cloud-Ressourcen und vor Ort betriebene Ressourcen werden in einfachen Textdateien beschrieben und diese Dateien in der Versionsverwaltung gehalten. Terraform ermittelt, welche realen Objekte es erstellen, ändern oder zerstören muss, damit die laufende Infrastruktur den Dateien entspricht. Das Programm selbst wird als `terraform` aufgerufen, kleingeschrieben, wie der Name in jedem Befehl und Skript erscheint.

Die Sprache ist deklarativ. HashiCorps Dokumentation beschreibt das als „ein angestrebtes Ziel zu beschreiben und nicht die Schritte, die dorthin führen“. Eine Konfiguration sagt, dass eine Maschine bestimmter Größe in einem bestimmten Netzwerk existiert. Sie zählt nicht die API-Aufrufe auf, die eine solche Maschine erzeugen würden. Dateien tragen die Erweiterung `.tf` ; daneben gibt es eine JSON-basierte Variante mit der Erweiterung `.tf.json` . Die native Syntax ist auf Grundlage einer Syntax namens HCL definiert und hat zwei Grundbausteine: einen Block als Behälter für weitere Inhalte und ein Argument, das einem Namen einen Wert zuweist.

Auf drei Grundgedanken baut das übrige Werkzeug auf.

Provider sind Plugins. Die Konfiguration sagt Terraform, welche Plugins zu installieren sind, und `terraform init` findet sie, lädt sie herunter und installiert sie. Laut HashiCorp ermöglichen Provider die Arbeit mit praktisch jeder Plattform oder jedem Dienst, der eine zugängliche API besitzt. Die öffentlich verfügbaren stehen in der Terraform Registry. Ein Provider definiert die Ressourcentypen seiner Plattform. Dasselbe Kernprogramm kann deshalb eine DNS-Zone, eine virtuelle Maschine und eine Überwachungsregel bei drei verschiedenen Anbietern verwalten.

Module sind Verzeichnisse. Ein Modul ist eine Sammlung von `.tf` - und `.tf.json` -Dateien in einem Verzeichnis. Das Root-Modul ist das Arbeitsverzeichnis, in dem Terraform aufgerufen wird.

Der Zustand ist das Gedächtnis. Terraform hält die reale Infrastruktur in einer Zustandsdatei fest, die seine Einführung als maßgebliche Quelle für die Umgebung bezeichnet. Der Zustand ordnet jede Ressource der Konfiguration dem realen Objekt zu, das Terraform verwaltet, samt Metadaten. Standardmäßig ist das eine lokale Datei namens `terraform.tfstate` ; die vorherige Version liegt daneben als `terraform.tfstate.backup` .

Die Lizenz hat sich geändert. Am 10. August 2023 kündigte HashiCorp an, die Quellcodelizenz aller künftigen Produktversionen von der Mozilla Public License 2.0 auf die Business Source License 1.1 umzustellen. APIs, SDKs und fast alle anderen Bibliotheken bleiben dagegen unter MPL 2.0. OpenTofu beschreibt sich als Terraform-Fork, den Gruntwork, Spacelift, Harness, Env0, Scalr und weitere Beteiligte angestoßen haben und der unter der Schirmherrschaft der Linux Foundation entwickelt wird. Sein Repository steht unter MPL 2.0. Die Website stellt es als Ersatz für Terraform vor, der bestehende Arbeitsabläufe und Konfigurationen beibehält.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist Terraform?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Die gesamte Einrichtung wird in Dateien festgehalten, damit sie sich exakt wiederherstellen lässt und nicht nur im Kopf einer einzelnen Person existiert.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 1.000 RBAC-Regeln für sichere Linux-Server — 1,000 Zugriffsregeln zum Schließen einer Sicherheitslücke

1.000 RBAC-Regeln für sichere Linux-Server Kontakt aufnehmen →

Warum es wichtig ist

HashiCorp beschreibt den grundlegenden Ablauf in drei Schritten. Write: Ressourcen definieren, die mehrere Provider und Dienste umfassen können. Plan: Terraform ermittelt, was es erstellen, ändern oder zerstören würde, und stellt das als Ausführungsplan dar. Apply: Nach der Freigabe führt es die Änderungen aus. Ihre Reihenfolge richtet sich nach den Abhängigkeiten zwischen Ressourcen, die es aus einem Ressourcengraphen ableitet.

Der mittlere Schritt trägt den größten Teil des Nutzens. Ein Plan ist eine Vorschau, die jemand, der die Änderung nicht geschrieben hat, lesen kann, bevor etwas geschieht. Ein Reviewer sieht mehr als die Dateiunterschiede: die Folgen, zwei Ressourcen erstellt, eine aktualisiert und hin und wieder eine zerstört, die niemand anfassen wollte. Diese letzte Zeile in einem Pull Request zu entdecken ist billig; sie erst danach zu entdecken ist sehr teuer.

Der zweite Grund ist die Reichweite. Weil das Wissen über jede Plattform in ihrem Provider liegt, decken eine Sprache, ein Arbeitsablauf und dieselben Review-Gewohnheiten alle Provider eines Teams ab. Ein Unternehmen mit zwei Clouds und einem separaten DNS-Host kann alle drei in einem Repository beschreiben und auf dieselbe Weise prüfen. Microsofts Bicep trifft in seinem Azure-Resource-Manager-Ablauf eine andere Abwägung: Es führt keine eigene Zustandsdatei, weil Azure den Bereitstellungszustand speichert. Das macht Bicep nicht zu einer reinen Azure-Sprache. Seine Erweiterungen unterstützen auch Ressourcen außerhalb von Resource Manager, darunter Kubernetes und Microsoft Graph.

Der dritte Grund ist der Zustand. Weil Terraform festhält, welches reale Objekt zu welchem Block gehört, kann es ein verwaltetes Objekt von einem unterscheiden, das es nie gesehen hat, und die Aufzeichnung mit dem laufenden Bestand vergleichen. Diese Aufzeichnung ist zugleich der Preis des Werkzeugs. Sie muss für das ganze Team erreichbar sein, braucht eine Sperre gegen zwei gleichzeitige Schreibvorgänge und kann Geheimnisse enthalten. Ihre Zugriffskontrolle muss deshalb genauso streng sein wie die der Geheimnisse selbst.

Es gibt auch kleinere Kosten. HCL ist eine weitere zu lernende Sprache, und jeder Provider hat eigene Ressourcentypen und Eigenheiten. Teams, die auf Terraform ein Produkt aufbauen, müssen außerdem eine Lizenz lesen: HashiCorps Lizenz-FAQ widmet der Frage, was als konkurrierendes Angebot gilt, einen eigenen Punkt. OpenTofu bietet denselben Ablauf unter MPL 2.0.

Wie es in der Praxis aussieht

HashiCorps Seite zur Ressourcensyntax verwendet diesen Block als Beispiel:

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

```
}
```

`resource` ist der Blocktyp und `aws_instance` der vom AWS-Provider definierte Ressourcentyp. `"web"` ist der lokale Name: Terraform verfolgt die Ressource unter dieser Kennzeichnung in seiner Zustandsdatei; die Einstellungen der realen Maschine beeinflusst sie nicht. Die beiden Zeilen im Block sind Argumente.

Drei Befehle führen von der Textdatei zur laufenden Maschine:

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

`terraform init` bereitet das Arbeitsverzeichnis vor: Es installiert die angeforderten Provider-Plugins und richtet den Ort ein, an dem der Zustand gespeichert wird. Der Befehl lässt sich gefahrlos mehrfach ausführen. Laut Dokumentation gehört die Lockdatei für Abhängigkeiten in die Versionsverwaltung.

`terraform plan -out=tfplan` berechnet den Plan und schreibt ihn in einem nicht direkt lesbaren Format in eine Datei. Die Planung wendet die vorgeschlagenen Infrastrukturänderungen nicht an. `terraform apply tfplan` führt dann die Operationen aus dieser Datei aus, ohne eine Bestätigung einzuholen: HashiCorp wertet die Übergabe eines gespeicherten Plans bereits als Freigabe. Aus demselben Grund wird das Flag `-auto-approve` ignoriert, wenn eine Plandatei übergeben wird.

Im Team ergibt sich die Trennung von selbst. Ein CI-Job erzeugt den Plan zum Pull Request und bewahrt die Datei auf. Eine Person liest Plan und Dateiunterschiede, und nach dem Merge wendet ein zweiter Job genau diese Datei an. Ausgeführt wird damit, was gelesen wurde. Die Apply-Dokumentation ergänzt eine leicht zu übersehende Bedingung: Niemand darf die Infrastruktur außerhalb des Terraform-Ablaufs ändern können. Sonst beschreibt der Plan eine Welt, die sich bereits weiterbewegt hat.

Auf dieser Website stellt der Technologieüberblick Terraform neben Ansible: Beide halten schriftlich fest, wie die Maschinen beschaffen sein sollen, damit eine Umgebung gelesen werden kann, statt rekonstruiert werden zu müssen. Ein

Portfolioeintrag setzt denselben Gedanken mit einem anderen Werkzeug um: Azure-Infrastruktur wurde mit Bicep als Code bereitgestellt. Test-, Staging- und Produktumgebung entstehen alle aus einem Satz von Vorlagen. Da diese Parameter annehmen, wird eine Umgebung durch einen Pull Request hinzugefügt oder geändert. Neben diesem Artikel zeigt der Eintrag die zuvor beschriebene Abwägung: Bicep hat keine eigene Zustandsdatei, weil Azure die Aufzeichnung führt. Terraform hält dagegen seinen eigenen Zustand und arbeitet dafür über mehr als eine Cloud hinweg.

Wo es schiefgeht

Die meisten Vorfälle mit Terraform haben denselben Ablauf: Die Konfiguration ist gültig, der Lauf gelingt, und etwas Reales ist verschwunden oder verändert.

Eine umbenannte Ressource wird zerstört. Wird die Kennzeichnung `"web"` in `"frontend"` geändert, hat die Ressource eine neue Adresse. Laut HashiCorps Refactoring-Anleitung deutet Terraform eine geänderte Adresse standardmäßig als Anweisung, das bestehende Objekt zu zerstören und ein neues anzulegen. Bei einer entbehrlichen Maschine kostet das ein paar Minuten, bei einer Datenbank die Daten. Die Lösung ist ein `moved`-Block, der die Adresse ohne Zerstörung aktualisiert:

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

Den Block nach getaner Arbeit behalten. HashiCorp empfiehlt ausdrücklich, historische `moved`-Blöcke beizubehalten, und bezeichnet das Entfernen eines solchen Blocks als inkompatible Änderung.

Zwei Personen wenden Änderungen mit zwei Kopien des lokalen Zustands an. In der Ausgangskonfiguration liegt der Zustand jedes Workspace in einer lokalen Datei. Für eine Person mit einem Laptop passt das. Sobald jemand anderes Terraform aus einer weiteren Kopie startet, widersprechen sich die beiden Zustandsdateien darüber, was existiert, und jeder Lauf handelt nach seiner eigenen Fassung. Die Lösung ist, den Zustand wie von HashiCorp empfohlen in HCP Terraform oder in einem entfernten Speicher mit Sperren und Zugriffskontrolle zu halten. Terraform sperrt den Zustand dann automatisch bei

jeder Operation, die ihn schreiben könnte, und fährt nicht fort, wenn es die Sperre nicht erhält. Nicht jede Speicheroption unterstützt Sperren. Das vor der Auswahl prüfen und `-lock=false` unangetastet lassen: Das Flag existiert, die Dokumentation rät aber davon ab.

Eine festhängende Sperre wird zwangsweise aufgehoben, während jemand noch arbeitet. Schlägt die automatische Entsperrung fehl, bleibt die Sperre bestehen. Der Befehl `force-unlock` löst sie mit der von Terraform ausgegebenen Sperr-ID. Die Dokumentation zum Sperren mahnt zu großer Vorsicht: Wird eine fremde Sperre gelöst, können zwei Prozesse denselben Zustand schreiben. Der Befehl ist nur für die eigene Sperre nach gescheiterter automatischer Entsperrung gedacht. Vor jedem erzwungenen Eingriff klären, wem die Sperre gehört und ob der zugehörige Lauf noch aktiv ist.

Die Zustandsdatei verrät Geheimnisse. HashiCorp warnt, dass Geheimnisse im Zustand landen können und bei Speicherung ohne sichere Zugriffskontrolle offengelegt werden können. Eine Zustandsdatei in einem öffentlich lesbaren Bucket oder als Kopie in einem Build-Artefakt ist ein Zugangsdatenleck. Sie aus dem Repository heraushalten, mit derselben Zugriffskontrolle versehen wie die enthaltenen Geheimnisse und nie von Hand bearbeiten. Für notwendige Korrekturen gibt es den Befehl `terraform state`.

Die Freigabe wird zur Formalität. Das Flag `-auto-approve` überspringt die interaktive Freigabe des Plans. Die Übergabe einer gespeicherten Plandatei tut dasselbe, weil die Datei als Freigabe zählt. Die Apply-Dokumentation warnt, dass das auch zerstörerische Änderungen umfasst, darunter das Löschen von Ressourcen. Ein CI-Job, der bei jedem Merge Änderungen ohne geprüften Plan anwendet, hat den einen Moment entfernt, in dem eine Person die bevorstehende Zerstörung hätte sehen können. Die Lösung ist der oben beschriebene Ablauf mit gespeichertem Plan und verbindlichem menschlichem Review zwischen Plan- und Apply-Job. Wird `-auto-approve` in der Automatisierung eingesetzt, rät HashiCorp selbst dazu, zuerst einen Plan zu erzeugen und zu prüfen.

Weiterführende Lektüre

- What is Terraform? — HashiCorps Einführung: Write, Plan, Apply, Provider und Zustand.
- terraform apply command — gespeicherte Pläne, `-auto-approve` und der Befehl in der Automatisierung.

- State Locking — automatische Sperren, nicht unterstützte Speicher und die Warnung zu `force-unlock`.
- Refactor modules — warum eine geänderte Adresse Zerstören und Neuanlegen bedeutet, und der `moved`-Block.
- HashiCorp adopts Business Source License — die Ankündigung vom August 2023 zum Wechsel von MPL 2.0 auf BSL 1.1.
- OpenTofu — der unter der Schirmherrschaft der Linux Foundation gepflegte Terraform-Fork.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Bereitstellungsweg, die Betriebsrisiken und die Bereitschaftsverantwortlichen. Wir können eine Änderung mit Prüfungen, einem Wiederherstellungsweg und klaren Zuständigkeiten abgrenzen.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist Terraform?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/terraform/