Engineer

Infrastruktur als Code

Infrastruktur mit versionierten Definitionsdateien verwalten

Was es ist

Infrastruktur als Code bedeutet, die Grundlagen eines Systems – Maschinen, Netzwerke, Firewallregeln, Load Balancer und DNS-Einträge – in Textdateien zu beschreiben, diese in der Versionsverwaltung zu halten und ein Werkzeug daraus die reale Umgebung aufbauen zu lassen. Die Dateien sind die maßgebliche Fassung. Eine Infrastrukturänderung beginnt als Dateiänderung, durchläuft dieselbe Prüfung wie Anwendungscode und erreicht das laufende System erst, wenn das Werkzeug sie anwendet. Der Begriff wird auch mit Bindestrichen als Infrastructure-as-Code geschrieben und mit IaC abgekürzt.

NISTs Special Publication 800-204C, eine DevSecOps-Anleitung für auf Microservices aufgebaute Anwendungen, beschreibt Infrastruktur als Code als den Code, der die Ressourcen bereitstellt und konfiguriert, auf denen eine Anwendung läuft, „auf wiederholbare und konsistente Weise“. Dieselbe Anleitung sagt, dass diese Definitionen mit denselben Werkzeugen wie der Anwendungsquellcode versioniert werden. Wiederholbarkeit ist der ganze Zweck. Eine aus Dateien erzeugbare Umgebung lässt sich erneut erzeugen: neben der ersten, nach einer Katastrophe oder an einem Dienstagnachmittag in einem Testkonto.

Definitionen gibt es in zwei Grundformen. Eine deklarative Definition beschreibt das Ergebnis, etwa einen Server bestimmter Größe in einem bestimmten Netzwerk mit einem offenen Port. Das Werkzeug entscheidet, was erstellt, geändert oder unverändert gelassen wird. Eine imperative Definition ist eine Schrittfolge: dies erstellen, jenes anschließen, dann den Port öffnen. SP 800-204C ordnet Infrastruktur als Code dem deklarativen Code zu: Das Team beschreibt seinen Bedarf, und ein Werkzeug macht daraus eine laufende Instanz. HashiCorp beschreibt die Terraform-Sprache ebenso: als Beschreibung eines gewünschten Ziels statt der Schritte dorthin. Der Unterschied zeigt sich beim

zweiten Lauf. Eine Schrittfolge hält fest, wie etwas einmal gemacht wurde. Eine Deklaration hält fest, was existieren soll, und genau das muss das Werkzeug wissen, wenn ein Teil davon bereits existiert.

Es gibt keine einzelne Sprache dafür. Laut SP 800-204C wird der Code je nach Werkzeug entweder in einer allgemeinen Skriptsprache wie Python, JavaScript oder TypeScript oder in einer werkzeugeigenen Konfigurationssprache wie HashiCorps HCL geschrieben. Diese kann mit einem Standardformat wie JSON kompatibel sein oder auch nicht. Terraforms Sprache gehört zur zweiten Art und steht in Dateien mit der Endung `.tf`. Azure akzeptiert Resource-Manager-Vorlagen in JSON oder in der kürzeren Sprache Bicep. Ansible schreibt Playbooks in YAML und erreicht Maschinen über SSH mit deren vorhandenen Betriebssystemzugangsdaten, ohne dort einen Agenten zu installieren.

Zwei Aufgaben unter demselben Begriff sollten getrennt werden. Provisionierung erstellt die Teile: virtuelle Maschine, Festplatte, Netzwerk, Firewall. Konfiguration richtet ein, was innerhalb einer bereits vorhandenen Maschine läuft: Pakete, Konten, Dienstdateien. Manche Werkzeuge neigen zur einen, andere zur anderen Aufgabe; die Grenze ist fließend. Beides wird dadurch zu Infrastruktur als Code, dass die Definition in einem Repository liegt und das Werkzeug das reale System daran ausrichtet.

Werkzeuge unterscheiden sich auch darin, wie sie sich merken, was sie gebaut haben. Terraform führt eine Zustandsdatei, die Ressourcen der Konfiguration den real erstellten Objekten zuordnet. Die Dokumentation nennt diesen Zustand die maßgebliche Quelle für die Umgebung. Bicep führt keine Zustandsdatei, weil Azure selbst erfasst, was bereitgestellt wurde. Der Speicherort dieser Aufzeichnung entscheidet, was passiert, wenn zwei Menschen das Werkzeug gleichzeitig ausführen oder wenn die Aufzeichnung verloren geht.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist Infrastruktur als Code?“ 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

Vor Infrastruktur als Code wurde jede Umgebung von Hand eingerichtet und danach von Hand gepatcht. Staging erhielt eine Korrektur, Produktion nicht; ein für einen Test geöffneter Port blieb offen; eine von drei Maschinen bekam ein neueres Paket. SP 800-204C nennt das Drift: Die vorhandene Konfiguration ergibt nicht mehr den beabsichtigten Betriebszustand. Die Anleitung benennt auch den üblichen Einstieg, eine Änderung in einer Webkonsole oder auf der Kommandozeile, die am Code vorbeigeht. Funktioniert eine Veröffentlichung in Staging und scheitert in Produktion, ist Drift ein häufiger Grund.

Infrastruktur als Code verlegt die maßgebliche Fassung aus dem laufenden System heraus. Das Team ändert die Definition, und das laufende System folgt. Niemand meldet sich an, um eine Firewallregel zu ändern; die Datei mit der Firewallbeschreibung wird geändert, und das Werkzeug führt die Änderung aus. Daraus folgen drei Dinge.

Der erste Vorteil sind die Prüfung und die Änderungshistorie. Eine Netzwerkänderung wird zu einem Diff in einem Pull Request, den eine zweite Person liest, bevor etwas geschieht. Monate später beantwortet das Commitprotokoll, wer einen Port geöffnet hat und warum, statt dass jemand sich erinnern muss. Eine Änderung zurückzunehmen bedeutet einen Revert und einen weiteren Werkzeuglauf.

Der zweite Vorteil ist Wiederholbarkeit. Eine gut geschriebene Definition ist idempotent: Auf eine bereits passende Umgebung angewendet ändert sie nichts; eine abgewichene bringt sie zurück. Dafür gibt es zwei Wege: das Vorhandene anpassen oder es verwerfen und aus der Definition neu aufbauen. Wird der zweite Weg zur Regel, ist das unveränderliche Infrastruktur.

Der dritte Vorteil sind günstige Umgebungen. SP 800-204C beschreibt Werkzeuge, die den gesamten Lebenszyklus von Bereitstellung bis Abbau automatisieren. Manche Infrastruktur lebt laut Anleitung zu kurz, als dass ihre Wartung lohnte. Ein Team kann bei Bedarf mehrere Testumgebungen erstellen und wieder entfernen. Auch der Code selbst lässt sich prüfen, bevor er etwas berührt: Die Anleitung verlangt, jeden Teil vor dem Zusammenführen auf Schwachstellen zu untersuchen. Ein Branch kann eine eigene kurzlebige Kopie der Produktionsstruktur erhalten, und ein Prüfer kann vor seiner Freigabe eine Vorschau der Änderung sehen.

Die Kosten sollten klar benannt werden. Eine Firewallregel über einen Pull Request zu ändern dauert länger als in einer Webkonsole, und bei einem Ausfall fühlt sich das teuer an. Jedes Werkzeug verlangt das Lernen einer eigenen Sprache und eines eigenen Zustandsmodells. Und eine Datei, die eine Umgebung aufbauen kann, kann sie auch löschen. Der Hebel reicht in beide Richtungen gleich weit. Deshalb ist die Vorschau so wichtig wie der anschließende Lauf.

Wie es in der Praxis aussieht

Hier durchläuft eine Änderung den Weg von der Bearbeitung bis zum laufenden System. In diesem Aufbau führt ein CI-Job das Werkzeug aus und veröffentlicht das Ergebnis seines Probelaufs im Pull Request:

```
Tag 1   09:10  Netzwerkdefinition ändern: TCP 443 aus dem Büro zulassen
        09:12  Pull Request öffnen; CI führt einen Probelauf aus (Plan)
        09:14  Plan im PR veröffentlicht: eine Regel hinzugefügt, keine entfernt
        10:30  Prüfer liest Plan und Diff und gibt den Plan frei
        10:31  Zusammenführen; CI wendet den geprüften Plan an
        10:33  Zweiter Lauf derselben Definition meldet keine Änderungen
Tag 40          Warum ist 443 offen? Commitprotokoll: wer und warum
```

Der Plan ist der Teil, der seinen Aufwand rechtfertigt. Das Diff zeigt die beabsichtigte Änderung; der Plan zeigt, was das Werkzeug gleich tun wird, einschließlich unerwarteter Dinge wie einer zur Zerstörung vorgesehenen Ressource. Terraform kann einen Plan mit `-out` in einer Datei speichern. Erhält `terraform apply` diese Datei, führt es die darin enthaltenen Operationen aus. Was um 10:31 läuft, ist damit das um 10:30 Genehmigte. Der stille zweite Lauf um 10:33 prüft, ob Definition und System jetzt übereinstimmen.

Auf dieser Website beschreibt die Dienstleistungsseite zu Infrastruktur als Code die wiederholbare Bereitstellung von Servern aus versionierter Konfiguration, statt sich auf jemandes Erinnerung zu verlassen.

Sicher dorthin zu kommen ist vor allem eine Frage der Reihenfolge. Ein Portfolioeintrag vom September 2026 zeigt warum. Er ergänzt einen zweiten Server bei einem anderen Anbieter. Die vorherige Maschine war von Hand eingerichtet und erst später geschützt worden. Ein frisch erstellter Host mit standardmäßiger Passwortrichtlinie stand daher zeitweise zum Internet offen. Diesmal wurde der Server aus dem Repository erstellt, und der Schutz war vor der Maschine vorhanden. Das Firewall-Play läuft vor dem Server-Play. Beim

zweiten Anbieter werden die Tags vor der Cloudfirewall angelegt, weil auf Tags gerichtete Regeln ohne vorhandene Tags nichts zum Anknüpfen haben. Eine Wiederholung schadet nicht, weil die erstellende Rolle eine schon vorhandene Maschine übernimmt. Sie ist die einzige Rolle, die Geld kostet. Deshalb hält ihre eigene Dokumentation Region, Maschinentyp und Ausstattung sowie den Monatspreis in Euro fest. Der Eintrag benennt auch offen, was ein Probelauf übersieht. Beide Firewalls werden über REST-API-Aufrufe der Anbieter aus einer allgemeinen HTTP-Aufgabe konfiguriert. Diese unterstützt den Check-Modus nicht und wird deshalb im Probelauf übersprungen. Die eigentliche Probe ist ein eigenständiger Lauf des Firewall-Plays, der nichts kostet und sich rückgängig machen lässt.

Wo es schiefgeht

Infrastrukturcode scheitert oft auf eine Weise, die nach Erfolg aussieht. Das Werkzeug beendet sich sauber, der CI-Lauf ist grün, und Dateien und reales System widersprechen sich unbemerkt.

Die nächste Ausführung nimmt eine während eines Vorfalls vorgenommene Handkorrektur zurück. Jemand öffnet um zwei Uhr morgens einen Port in der Konsole, der Ausfall endet, und eine Woche später schließt ihn der Routinelauf wieder, weil die Dateien nie davon erfuhren. Auch das Gegenteil kommt vor: Niemand führt das Werkzeug erneut aus, und die Dateien beschreiben die Realität nicht mehr. Übernehmen Sie die Notfalländerung noch am selben Tag in den Code und führen Sie nach Zeitplan einen Vergleich von Dateien und laufendem System aus. SP 800-204C verlangt ihn unmittelbar nach jeder Bereitstellung und in Abständen während des Betriebs. Terraforms detaillierter Exitcode des Planbefehls macht daraus einen kurzen nächtlichen Job:

```
terraform plan -detailed-exitcode
# exit 0: no changes, exit 1: error, exit 2: changes present
```

Auf einem bereits angewendeten Branch bedeutet Exitcode 2, dass das reale System vom Code abgewichen ist. Jemand sollte den Grund ermitteln.

Eine aus der Datei entfernte Ressource wird nicht immer gelöscht. Werkzeuge sind sich uneinig, was eine fehlende Ressource bedeutet. Für Azure-Resource-Manager-Vorlagen sagt Microsofts What-if-Dokumentation, dass eine vorhandene, aber nicht in der Vorlage enthaltene Ressource nur im vollständigen

Modus gelöscht wird. In jedem anderen Modus läuft sie weiter und verursacht weiter Kosten. Lernen Sie vorab die Löschregel Ihres Werkzeugs kennen und lesen Sie die Vorschau sowohl auf erwartete als auch auf befürchtete Löschungen.

Der Probelauf sieht nicht alles. Eine Vorschau ist ein Modell der Änderung, keine Garantie. Microsofts What-if-Dokumentation nennt Eigenschaften, die als gelöscht gemeldet werden, obwohl Azure bei der Bereitstellung nur einen Standardwert setzt, und bezeichnet das Ergebnis als Rauschen. Jede über die Funktion `reference` gesetzte Eigenschaft erscheint bei jedem Lauf als Änderung. Der Portfolioeintrag oben zeigt die andere Lücke: Eine Aufgabe ohne Check-Modus-Unterstützung wird beim Probelauf übersprungen; die Vorschau sagt daher nichts darüber. Machen Sie sich mit den bekannten irreführenden Vorschauangaben Ihres Werkzeugs vertraut und behandeln Sie alles, was die Vorschau nicht sieht, als ungetestet, bis eine Probe an etwas Günstigem und Umkehrbarem es abdeckt.

Geheimnisse landen im Repository oder im Zustand. Ein Passwort in einer Definitionsdatei bleibt in der Geschichte jedes Klons. Weniger offensichtlich ist Terraforms Warnung, dass sein Zustand Geheimnisse enthalten kann. Standardmäßig ist das eine lokale Datei namens `terraform.tfstate`. Halten Sie Geheimnisse in einem dafür vorgesehenen Speicher und verweisen Sie über ihren Namen darauf. Speichern Sie den Zustand wie von HashiCorp empfohlen remote mit Sperren und Zugriffskontrolle und bearbeiten Sie die Zustandsdatei niemals von Hand. Für nötige Änderungen gibt es `terraform state`.

Ein Skript wird als Definition verkleidet. Ein Shellskript im Repository, das einen Server erstellt, ist versioniert, aber nicht idempotent: Zweimal ausgeführt erzeugt es zwei Server oder einen Fehler auf halber Strecke. Verwenden Sie deklarative Ressourcen oder Schritte, die vor dem Handeln prüfen, wie die Rolle oben, die eine vorhandene Maschine übernimmt. Ansibles Dokumentation setzt den Maßstab klar: Ist ein System schon in dem vom Playbook beschriebenen Zustand, ändert eine erneute Ausführung nichts.

Weiterführende Lektüre

- DevSecOps für eine Microservices-Anwendung mit Service Mesh umsetzen (NIST SP 800-204C) — Infrastruktur als deklarativer, versionierter Code, Drift und Schwachstellenprüfung.

- Dokumentation der Terraform-Sprache — Die deklarative Sprache, ihre Ressourcen und ihre Blöcke.
- Befehl plan — Probelauf, gespeicherte Pläne und detaillierte Exitcodes.
- Zustand — Was Terraform aufzeichnet, wo es die Daten speichert und warum sie sensibel sind.
- Einführung in Ansible — Agentenlose Automatisierung über SSH, Playbooks und Sollzustand.
- What-if für Vorlagenbereitstellungen — Vorschau einer Azure-Bereitstellung, Änderungsarten und bekanntes Rauschen.

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 Infrastruktur als Code?“ 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/infrastructure-as-code/