Engineer

Unveränderliche Infrastruktur

❄ Server ersetzen, statt vorhandene Server zu verändern

Was es ist

Unveränderliche Infrastruktur ist eine Betriebsweise für Server, bei der eine einmal in Betrieb genommene Maschine nie verändert wird. HashiCorps Well-Architected Framework beschreibt sie als Infrastruktur, die „nach der Bereitstellung niemals verändert, sondern nur ersetzt wird“. Eine neue Anwendungsversion, eine aktualisierte Bibliothek, ein Sicherheitspatch und eine geänderte Einstellung kommen alle auf demselben Weg an: Eine neue Maschine oder ein neuer Container wird aus einem neuen Image gebaut und tritt an die Stelle des Vorgängers. Dieser wird anschließend abgeschaltet und gelöscht.

Das Gegenmodell ist veränderliche Infrastruktur: Ein langlebiger Server wird an Ort und Stelle aktualisiert. Im Juni 2013 beschrieb Chad Fowler, was nach Jahren solcher Aktualisierungen aus einem Server wird: immer wieder gepatcht, mit während eines Ausfalls von Hand geänderten Konfigurationen und Cronjobs, von denen nur eine Person weiß, bis das Team sowohl seine Veränderung als auch seinen Ersatz fürchtet. Seine Lösung, die er unveränderliche Bereitstellungen nannte, war, jedes System automatisiert aufzubauen und danach unberührt zu lassen. Ein Upgrade oder eine neue Veröffentlichung bedeutet dann, einen frischen Server oder ein neues Image zu bauen und den Vorgänger wegzuwerfen.

Zwei Arten von Images machen den Ansatz praktikabel. Ein Maschinenimage ist nach der Packer-Dokumentation eine einzelne statische Einheit mit einem vorkonfigurierten Betriebssystem und der installierten Software. Das Community-Werkzeug Packer baut solche Images: Aus einer Quellkonfiguration erzeugt es entsprechende Images für mehrere Plattformen. Containerimages sind die zweite Art, und Docker formuliert es klar: Ein erstelltes Image kann nicht verändert werden. Es bleibt nur, ein neues Image zu erstellen oder Änderungen als neue Schichten daraufzulegen.

Der Begriff bezeichnet eine Betriebsweise, kein Produkt, das man installiert. Er hat auch eine Grenze. Anwendungsserver lassen sich bei jeder Veröffentlichung ersetzen, aber ihre Daten müssen irgendwo liegen, wo sie jeden Austausch überdauern. An dieser Grenze fällt der größte Teil der praktischen Entwurfsarbeit an.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist unveränderliche Infrastruktur?“ 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

Der erste Grund ist Drift. HashiCorps Framework benennt sie direkt: Manuelle Updates und Änderungen am laufenden System lassen Server mit der Zeit von ihrer ursprünglichen Konfiguration abweichen. Zwei Server, die mit einem Jahr Abstand aus derselben Ausgangskonfiguration gestartet wurden, müssen am Ende nicht identisch sein, weil jeder eine andere Folge aus Korrekturen, Wiederholungsversuchen und Handänderungen erhalten hat. Ist jeder Server eine frische Kopie eines Images, beschreibt das Image den Server vollständig; es bleibt nichts, das abweichen könnte.

Der zweite Grund ist die Rückkehr zur Vorgängerversion. Verhält sich eine Veröffentlichung falsch, empfiehlt das Framework, das vorherige Artefakt erneut bereitzustellen, statt die laufenden Server zu reparieren. Das vorherige Image ist ein bekannter Zustand, der bereits in Produktion lief. Das Team muss daher nicht unter Druck aus Erinnerung und Shellverlauf rekonstruieren, wie der Server gestern aussah.

Der dritte Grund sind weniger manuelle Eingriffe in Produktion. Das Framework weist darauf hin, dass Anmeldungen an Produktionsservern für manuelle Updates menschliche Fehler wie Tippfehler und vergessene Schritte einführen. Bei unveränderlicher Infrastruktur muss sich überhaupt niemand von Hand an einem Server anmelden. Weniger Menschen brauchen überhaupt einen Zugang zur laufenden Maschine, und wer einen hat, muss ihn selten nutzen.

Der vierte Grund ist die Prüfspur. Auditoren wollen laut derselben Seite Belege dafür, dass Produktion nicht außerhalb genehmigter Verfahren verändert wurde. Manuelle Änderungen hinterlassen aber unvollständige Aufzeichnungen. Ist jede Änderung ein neues Artefakt, zeigen Versionsverwaltung und CI/CD-Protokolle, was wann geändert wurde und wer es genehmigt hat. Ein Sicherheitspatch folgt demselben Weg wie eine Funktionsveröffentlichung: Er wird zu einem neuen Artefakt, das das laufende ersetzt, und deshalb genauso gebaut, getestet und dokumentiert wird.

Nichts davon ist kostenlos. Das Framework hält fest, dass der Start einer virtuellen Maschine länger dauert und mehr Ressourcen verbraucht als der eines Containers. Container brauchen wiederum einen Orchestrator für ihren Betrieb. Jede Änderung, selbst eine einzeilige Einstellung, bedeutet nun einen vollständigen Build und einen Austausch. Der Build muss deshalb automatisch und schnell genug sein, damit niemand versucht ist, ihn zu überspringen. Zustand muss vor dem ersten Austausch von den ersetzbaren Maschinen ausgelagert werden, nicht danach.

Wie es in der Praxis aussieht

Eine Packer-Vorlage beschreibt den Bau eines Maschinenimages. Diese basiert auf dem Beispiel aus der Dokumentation des Amazon-EBS-Builders von Packer. Ergänzt wurde ein `build` -Block in der Form aus der Referenz zum Buildblock:

```
source "amazon-ebs" "web" {
  access_key    = var.aws_access_key
  secret_key    = var.aws_secret_key
  region        = "us-east-1"
  source_ami    = "ami-fce3c696"
  instance_type = "t2.micro"
  ssh_username  = "ubuntu"
  ami_name      = "web {{timestamp}}"
}

build {
  sources = ["source.amazon-ebs.web"]

  provisioner "shell" {
    scripts = fileset(".", "scripts/{install,secure}.sh")
  }
}
```

Der Block `source` benennt den Builder und das Basisimage `source_ami`, dessen Rootvolume für eine laufende Instanz kopiert und dort eingerichtet wird. Der Block `build` gibt an, welche Quellen gebaut und wie sie eingerichtet werden. Der Shell-Provisioner führt die beiden Skripte innerhalb dieser laufenden Maschine aus, bevor sie zum statischen Artefakt wird. Dieses Artefakt ist ein neues Image. Die Deklarationen der beiden Variablen sind hier ausgelassen. Ein Befehl führt jeden Build der Vorlage aus und erzeugt die Artefakte:

```
packer build web.pkr.hcl
```

Die Bereitstellung ist die zweite Hälfte. HashiCorps Framework stellt die beiden Werkzeuge nebeneinander: Images werden mit Packer gebaut, mit Terraform bereitgestellt und bei einem Update vollständig ersetzt. Kann eine Änderung nicht am vorhandenen Objekt erfolgen, zerstört Terraform laut seiner Lifecycle-Referenz standardmäßig zuerst das bestehende Objekt und erstellt dann den Ersatz. Updates in diesem Ablauf erfordern keine Anmeldung an einem laufenden Produktionsserver. Eine Skriptänderung erzeugt ein neues Image, dieses erzeugt neue Maschinen, und die alten werden entfernt.

Dieselbe Disziplin funktioniert auch auf einem einzigen Host. Der Portfolioeintrag zum Betrieb von Containern mit Podman und Quadlet trennt die Einrichtung der Umgebung von der Anwendungsbereitstellung. Root bereitet den Bereitstellungspfad einmalig vor. Danach veröffentlicht ein normaler Nutzer ohne Sonderrechte neue Versionen; er darf genau ein Skript ausführen und sonst nichts. Eine Veröffentlichung besteht aus einem frischen Image und anschließendem Dienstneustart, ohne Rootshell oder beliebige Befehle. Der Eintrag zu CI/CD mit GitHub Actions geht beim Image weiter: Das Produktionsfrontend läuft auf einer Distroless-Basis ohne Shell und Paketmanager. Ein Build wird durch Workflows von einer Umgebung in die nächste befördert, nicht durch eine Person. Ein solches Image bietet kaum noch eine Möglichkeit zur Anmeldung und kein Werkzeug, um etwas zu installieren. Seinen Nachfolger zu bauen ist damit der einzig vernünftige Änderungsweg.

Der Eintrag zu von Grund auf wiederaufbaubaren Servern ist ein sorgfältiges Beispiel für das andere Modell. Ansible bringt dort einen laufenden Host in den im Repository beschriebenen Zustand. Jedes Play ist idempotent, und das Repository liefert die Konfiguration zum Neuaufbau der deklarierten Umgebung. Der Eintrag benennt offen die Kosten einer Änderung am bestehenden System. Jede Änderung dauert länger als eine direkte Bearbeitung auf dem Server. Ein

auf halber Strecke abbrechender Lauf hinterlässt den Host in einem schlechteren Zustand als ein Lauf, der den Start verweigert; deshalb läuft heute ein Preflight-Play voraus. Ein halb veränderter Host ist ein Schritt zu dem fragilen Server, den Fowler beschrieb. Die Maschine zu ersetzen, statt sie zu ändern, ist der andere Weg um dieses Risiko herum.

Wo es schiefgeht

Das Modell ist einfach zu beschreiben und leicht auszuhöhlen. Die meisten Fehler entstehen aus einer Ausnahme, die zunächst harmlos schien.

Eine schnelle Korrektur über SSH bringt die Drift zurück. Unter Druck meldet sich jemand am laufenden Server an und bearbeitet eine Datei. Die Korrektur wirkt, bis die nächste Bereitstellung die Maschine ersetzt. Dann verschwindet sie samt jeder Aufzeichnung, warum sie nötig war. Bis dahin entspricht der laufende Server nicht mehr seinem Image, genau der Zustand, den der Ansatz verhindern soll. Leiten Sie jede Änderung, auch die dringende, durch den Imagebuild und halten Sie diesen schnell genug, um der schnellste Weg zu sein. HashiCorps Anleitung schließt die Anmeldung am laufenden Server für Handänderungen oder Updates vollständig aus.

Auf die Maschine geschriebene Daten werden mit ihr gelöscht. Laut Docker-Dokumentation werden beim Löschen eines Containers auch die darin vorgenommenen Dateiänderungen gelöscht. Uploads, erzeugte Dateien oder eine lokale Datenbank auf einer ersetzbaren Maschine verschwinden bei der nächsten Veröffentlichung. Halten Sie Zustand außerhalb des ersetzten Teils. Docker-Volumes bewahren Daten über den Lebenszyklus eines einzelnen Containers hinaus. Ein neuer Container am selben Volume findet die Dateien weiter vor. Bei Maschinen übernimmt eine separate Festplatte oder ein nicht zum Image gehörender Datenbankdienst diese Rolle.

Der Austausch legt den Dienst still. Terraform zerstört standardmäßig zuerst das vorhandene Objekt und erstellt danach den Ersatz. Dadurch gibt es ein Zeitfenster, in dem nichts läuft. Die Einstellung `create_before_destroy` im Block `lifecycle` einer Ressource kehrt die Reihenfolge um:

```
resource "aws_instance" "web" {
  # ...
  lifecycle {
```

```
    create_before_destroy = true
  }
}
```

HashiCorp verlangt hierfür eine ausdrückliche Aktivierung, weil zwei Kopien eines Objekts nicht immer nebeneinander existieren können: Viele Objekttypen verlangen einen eindeutigen Namen oder haben andere Grenzen. Die Referenz fordert dazu auf, die Einschränkungen jedes Ressourcentyps kennenzulernen, bevor man sich auf die Einstellung verlässt.

Ein Neubau beginnt unbemerkt mit einer neueren Basis. Statt eines festen `source_ami` kann der Amazon-EBS-Builder dieses über einen `source_ami_filter` ermitteln. Steht `most_recent` auf true, wählt der Filter das neueste passende Image, was laut Dokumentation für einen täglichen Distributionsbuild am nützlichsten ist. Das erleichtert es, aktuell zu bleiben. Zwei Builds derselben Vorlage an verschiedenen Tagen beginnen dann aber mit unterschiedlichen Basen, und der Neubau einer alten Veröffentlichung reproduziert sie nicht mehr. Fixieren Sie für Veröffentlichungsimages `source_ami` auf ein bestimmtes Image und aktualisieren Sie diese Festlegung bewusst in einem geprüften Commit.

Geheimnisse werden ins Image eingebaut. HashiCorps Framework ist eindeutig: Artefakte dürfen niemals Geheimnisse wie Datenbankpasswörter, API-Schlüssel oder Verschlüsselungsschlüssel enthalten. Wer das Artefakt lesen kann, kann sie extrahieren. Eine Zugangsdatenänderung erzwingt dann den Neubau und die erneute Bereitstellung jedes Artefakts. Rufen Sie Geheimnisse erst beim Start der Maschine oder des Containers ab. HashiCorps eigenes Muster verwendet Vault: Eine startende Maschine authentifiziert sich und erhält temporäre, ablaufende Zugangsdaten. Zugangsdaten werden in Vault gewechselt, ohne etwas neu zu bauen. Die Vorlage oben folgt für ihre eigenen Schlüssel derselben Regel: Sie kommen als Variablen statt als Text in der Datei.

Das vorherige Artefakt fehlt, wenn es gebraucht wird. Rollback bedeutet, das vorherige Artefakt erneut bereitzustellen, und setzt dessen Fortbestand voraus. Ein Bereinigungsjob, der alte Images löscht, oder die Gewohnheit, für jeden Build denselben Imagenamen wiederzuverwenden, macht daraus einen Neubau aus altem Quellcode auf einer womöglich inzwischen veränderten Basis. Behalten Sie die Images der jüngsten in Produktion gelaufenen Veröffentlichungen, geben Sie jedem einen nie wiederverwendeten Namen und proben Sie die Rückkehr zu einem davon, bevor es darauf ankommt.

Weiterführende Lektüre

- Unveränderliche Infrastruktur erstellen — Das Well-Architected Framework zu Drift, Rollback, Imagearten und Geheimnissen.
- Server wegwerfen und Code verbrennen: unveränderliche Infrastruktur und austauschbare Komponenten — Chad Fowlers Beitrag vom Juni 2013, gelesen über eine Kopie im Internet Archive.
- Einführung in Packer — Was Packer baut und was ein Maschinenimage enthält.
- Amazon-EBS-Builder — Der oben verwendete Builder mit `source_ami` und `source_ami_filter`.
- Referenz zum Metaargument lifecycle — Terraforms Standardreihenfolge Zerstören–Erstellen und `create_before_destroy`.
- Containerdaten dauerhaft speichern — Warum Dateien im Container mit ihm verschwinden und wie Volumes sie erhalten.

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 unveränderliche Infrastruktur?“ 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/immutable-infrastructure/