Engineer

Terraform

інструмент HashiCorp, що планує й застосовує інфраструктуру

Що це таке

Terraform — це інструмент HashiCorp для інфраструктури як коду. Ресурси в хмарі та у власному дата-центрі описують у звичайних текстових файлах, зберігають ці файли в системі контролю версій, а Terraform сам визначає, які справжні об'єкти треба створити, змінити чи знищити, щоб робоче середовище відповідало файлам. Саму програму викликають як `terraform`, малими літерами, і саме так назва записана в кожній команді та скрипті.

Мова декларативна. Документація HashiCorp описує її як мову, що задає мету, а не кроки, які до неї ведуть. Конфігурація каже, що в певній мережі існує машина певного розміру; вона не перелічує викликів API, потрібних, щоб таку машину створити. Файли мають розширення `.tf`, а ще є варіант на основі JSON із розширенням `.tf.json`. Рідний синтаксис визначено через синтаксис під назвою HCL, і він має два основні елементи: блок, тобто контейнер для іншого вмісту, та аргумент, який надає назві значення.

На трьох ідеях тримається решта інструмента.

Провайдери — це плагіни. Конфігурація підказує Terraform, які плагіни встановити, а `terraform init` знаходить, завантажує й установлює їх. HashiCorp пише, що провайдери дають Terraform змогу працювати практично з будь-якою платформою чи сервісом, які мають доступний API, а загальнодоступні провайдери зібрані в Terraform Registry. Провайдер визначає типи ресурсів для своєї платформи, тож та сама основна програма може керувати зоною DNS, віртуальною машиною й правилом моніторингу в трьох різних постачальників.

Модулі — це каталоги. Модуль — це набір файлів `.tf` і `.tf.json`, що лежать разом в одному каталозі, а кореневий модуль — це робочий каталог, з якого запускають Terraform.

Стан — це пам'ять. Terraform відстежує справжню інфраструктуру у файлі стану, який вступ до документації називає джерелом істини для середовища. Цей файл зіставляє кожен ресурс конфігурації зі справжнім об'єктом, яким керує, і зберігає поруч метадані. За замовчуванням це локальний файл `terraform.tfstate`, а попередня версія лежить поруч як `terraform.tfstate.backup`.

Ліцензія змінилася. 10 серпня 2023 року HashiCorp оголосила, що переводить свій вихідний код з Mozilla Public License 2.0 на Business Source License 1.1 для всіх майбутніх випусків своїх продуктів, тоді як API, SDK і майже всі інші бібліотеки лишаються під MPL 2.0. OpenTofu називає себе форком Terraform, започаткованим Gruntwork, Spacelift, Harness, Env0, Scalr та іншими, який розвивається під опікою Linux Foundation. Репозиторій проєкту опубліковано під MPL 2.0, а сайт представляє OpenTofu як заміну Terraform, що зберігає наявні робочі процеси й конфігурації.

Застосуйте поняття до рішення

Застосуйте «Що таке Terraform?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Уся конфігурація записана у файлах, тож її можна відтворити точно, і вона не живе лише в чиїйсь голові.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Написала 1 000 правил RBAC для Linux-серверів — 1,000 правил доступу написано, щоб усунути вразливість

Написала 1 000 правил RBAC для Linux-серверів Зв'язатися з нами →

Чому це важливо

HashiCorp описує основний робочий процес трьома кроками. Write: ви визначаєте ресурси, які можуть охоплювати кількох провайдерів і сервіси. Plan: Terraform складає план виконання, де описано, що він створить, оновить або знищить. Apply: після схвалення Terraform виконує заплановані операції в правильному порядку з урахуванням залежностей між ресурсами, які він визначає, будуючи граф ресурсів.

Найбільше користі дає середній крок. План — це попередній перегляд, який може прочитати людина, що не писала зміну, ще до того, як щось станеться. Рецензент бачить більше, ніж різницю між файлами: він бачить наслідки, тобто два ресурси створено, один оновлено і час від часу один знищено, хоча чіпати його ніхто не збирався. Цей останній рядок дешево помітити в pull request і дуже дорого виявити згодом.

Друга причина — охоплення. Оскільки знання про кожну платформу живе в її провайдері, одна мова, один робочий процес і одні звички рев'ю охоплюють усіх провайдерів, якими користується команда. Компанія з двома хмарами й окремим постачальником DNS може описати всі три в одному репозиторії та перевіряти їх однаково. Вісер від Microsoft робить інший вибір у робочому процесі Azure Resource Manager: він не веде власного файлу стану, бо Azure зберігає стан розгортання. Це не означає, що Вісер працює лише з Azure: його розширення підтримують також ресурси поза Resource Manager, зокрема Kubernetes і Microsoft Graph.

Третя причина — стан. Оскільки Terraform записує, який справжній об'єкт належить якому блоку, він може відрізнити об'єкт, яким керує, від того, якого ніколи не бачив, і може порівняти свій облік із тим, що працює. Цей облік водночас і ціна інструмента. Він має лежати там, де до нього дістанеться вся команда, йому потрібне блокування, щоб два запуски не писали в нього одночасно, і в ньому можуть опинитися секрети, тож контроль доступу до нього має бути таким самим суворим, як до них.

Є й менші витрати. HCL — ще одна мова, яку треба вивчити, а кожен провайдер має власні типи ресурсів і примхи. Командам, що будують продукт поверх Terraform, варто ще й прочитати ліцензію: у FAQ HashiCorp про ліцензію є окреме питання про те, що вважають конкурентною пропозицією. OpenTofu пропонує той самий робочий процес під MPL 2.0.

Як це виглядає на практиці

Сторінка HashiCorp про синтаксис ресурсів наводить як приклад такий блок:

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

`resource` — це тип блоку, а `aws_instance` — тип ресурсу, який визначає провайдер AWS. `"web"` — локальна назва: за цією міткою Terraform відстежує ресурс у файлі стану, і на налаштування справжньої машини вона не впливає. Два рядки всередині блоку — це аргументи.

Три команди переводять файл із тексту в працюючу машину:

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

`terraform init` готує робочий каталог: установлює плагіни провайдерів, яких потребує конфігурація, і налаштовує місце, де зберігатиметься стан. Цю команду безпечно запускати кілька разів, а її документація радить комітити файл блокування залежностей у систему контролю версій.

`terraform plan -out=tfplan` обчислює план і записує його у файл у непрозорому форматі. Планування не застосовує запропонованих змін інфраструктури. Потім `terraform apply tfplan` виконує операції з цього файлу, не питаючи підтвердження, бо HashiCorp вважає саму передачу збереженого плану схваленням. З тієї ж причини прапорець `-auto-approve` ігнорується, коли передано файл плану.

У команді поділ складається сам собою. CI-завдання запускає plan на pull request і зберігає файл, людина читає план, а не лише різницю у файлах, а після злиття друге завдання застосовує саме цей файл, тож виконується те, що прочитали. Документація apply додає умову, про яку легко забути: подбайте, щоб ніхто не міг змінити інфраструктуру в обхід робочого процесу Terraform, інакше план описуватиме світ, який уже зрушив.

На цьому сайті сторінка про технології ставить Terraform поряд з Ansible: обидва записують бажаний стан машин, тож середовище можна прочитати, а не відновлювати навпомацки. Один запис у портфоліо застосовує ту саму ідею з іншим інструментом: інфраструктуру Azure розгорнули як код на Bicep. Середовища testing, staging і product збираються з одного набору шаблонів, а оскільки шаблони приймають параметри, нове чи змінене середовище — це pull request. Поряд із цією статтею приклад показує той

самий компроміс, що й вище. Вісер не має власного файлу стану, бо облік веде сама Azure, тоді як Terraform зберігає власний стан і натомість працює з різними хмарами.

Де виникають проблеми

Більшість аварій із Terraform мають однакову форму: конфігурація правильна, запуск успішний, а щось справжнє зникло або змінилося.

Перейменований ресурс знищується. Змініть мітку `"web"` на `"frontend"`, і ресурс отримає нову адресу. Посібник HashiCorp з рефакторингу каже, що за замовчуванням Terraform сприймає змінену адресу як вказівку знищити наявний об'єкт і створити новий. Для машини, яку не шкода викинути, це кілька хвилин; для бази даних це втрачені дані. Вихід — блок `moved`, який оновлює адресу, нічого не знищуючи:

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

Не прибирайте блок, коли він зробив свою справу. HashiCorp наполегливо радить зберігати старі блоки `moved` і пише, що видалення такого блоку ламає сумісність.

Двоє людей запускають apply з двома копіями локального стану. Від початку стан кожного workspace — це локальний файл. Це годиться для однієї людини на одному ноутбуці. Щойно колега запускає Terraform з іншої копії, два файли стану розходяться в тому, що існує, і кожен запуск діє за своєю версією. Вихід — тримати стан у HCP Terraform або у віддаленому сховищі з блокуванням і контролем доступу, як радить HashiCorp. Тоді Terraform автоматично блокує стан для кожної операції, що може в нього писати, і не продовжує, якщо блокування взяти не вдалося. Блокування підтримують не всі сховища, тож перевірте це до вибору, а прапорець `-lock=false` не чіпайте: він існує, але документація його не радить.

Завислий замок зламують, поки хтось іще працює. Коли автоматичне розблокування не спрацьовує, замок лишається, і команда `force-unlock` знімає його за ідентифікатором блокування, який вивів Terraform.

Документація про блокування просить бути дуже обережними: якщо зламати замок, який тримає хтось інший, у той самий стан можуть писати двоє, а команда призначена лише для власного замка після невдалого автоматичного розблокування. Вихід — з'ясувати, чий це замок і чи той запуск іще живий, перш ніж щось форсувати.

Файл стану розкриває секрети. HashiCorp попереджає, що секрети можуть потрапити в стан і що їх можна розкрити, якщо стан лежить у системі без надійного контролю доступу. Файл стану в бакеті, який може прочитати будь-хто, або скопійований в артефакт збірки, — це витік облікових даних. Тримайте його поза репозиторієм, давайте йому такий самий контроль доступу, як секретам, що в ньому лежать, і ніколи не правте його вручну: для потрібних виправлень існує команда `terraform state`.

Схвалення перетворюється на формальність. Прапорець `-auto-approve` пропускає інтерактивне схвалення плану, і збережений файл плану теж його пропускає, бо сам файл вважається схваленням. Документація apply наголошує, що це стосується й руйнівних операцій, як-от видалення ресурсів. CI-завдання, яке виконує apply після кожного злиття без переглянутого плану, прибрало єдину мить, коли людина могла б побачити знищення заздалегідь. Вихід — описаний вище процес зі збереженим планом і обов'язковим рев'ю людиною між завданням plan і завданням apply. Там, де `-auto-approve` використовують в автоматизації, HashiCorp сама радить спершу виконати plan і переглянути його.

Що почитати далі

- What is Terraform? — вступ від HashiCorp: write, plan, apply, провайдери й стан.
- terraform apply command — збережені плани, `-auto-approve` і запуск команди в автоматизації.
- State Locking — автоматичні блокування, сховища без них і застереження про `force-unlock`.
- Refactor modules — чому змінена адреса означає знищення й створення, і блок `moved`.
- HashiCorp adopts Business Source License — оголошення серпня 2023 року про перехід з MPL 2.0 на BSL 1.1.
- OpenTofu — форк Terraform, який розвивається під опікою Linux Foundation.

Словник

Обирайте за фактичними умовами

Опишіть шлях розгортання, експлуатаційні ризики та чергову команду. Ми можемо визначити зміну з перевірками, шляхом відновлення та чіткою відповідальністю.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке Terraform?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/terraform/