Engineer

незмінна інфраструктура

❄ заміна серверів новими замість змін на місці

Що це таке

Незмінна інфраструктура — це спосіб експлуатувати сервери, за якого машину, щойно її введено в роботу, більше ніколи не змінюють. В англомовних джерелах її називають immutable infrastructure. HashiCorp Well-Architected Framework описує її як інфраструктуру, яку після розгортання ніколи не модифікують, а лише замінюють. Нова версія застосунку, оновлена бібліотека, латка безпеки й змінене налаштування приходять однаково: як нова машина чи контейнер, зібрані з нового образу, що стають на місце старих. Після цього стару машину вимикають і видаляють.

Протилежність — змінна інфраструктура, де сервер живе довго й оновлюється там, де стоїть. У червні 2013 року Чед Фаулер описав, на що перетворюється такий сервер після років оновлень: його латають знову й знову, конфігурацію правлять уручну посеред аварії, на ньому працюють завдання cron, про які знає лише одна людина, і зрештою команда боїться і змінювати його, і замінювати. Як вихід Фаулер запропонував те, що назвав immutable deployments: будувати кожну систему автоматично й після створення більше її не чіпати. Оновлення чи нова версія тоді означає зібрати свіжий сервер або образ, а старий викинути.

Два види образів роблять цей підхід практичним. Машинний образ, за документацією Packer, — це єдина статична одиниця, що містить попередньо налаштовану операційну систему й установлене програмне забезпечення. Сам Packer — інструмент спільноти, який створює однакові машинні образи для кількох платформ з однієї вихідної конфігурації. Образи контейнерів — другий вид, і документація Docker каже про них прямо: створений образ змінити не можна, можна лише зробити новий образ або додати зміни поверх нього новими шарами.

Термін описує спосіб експлуатації системи, а не продукт, який встановлюють. І він має межу. Сервери застосунку можна замінювати з кожним релізом, але дані, з якими вони працюють, мусять жити десь, що переживе кожну заміну, і саме на цій межі зосереджена більша частина практичної проєктної роботи.

Застосуйте поняття до рішення

Застосуйте «Що таке незмінна інфраструктура?» 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

Перша причина — дрейф. Фреймворк HashiCorp називає його прямо: ручні оновлення й зміни на місці з часом віддаляють сервери від їхньої початкової конфігурації. Два сервери, налаштовані однаково з різницею в рік, не обов’язково залишаться ідентичними, бо кожен отримав свою послідовність виправлень, повторних спроб і ручних правок. Коли кожен сервер — свіжа копія одного образу, образ і є повним описом сервера, і дрейфувати більше нема чому.

Друга причина — відкат. Якщо реліз поводиться неправильно, фреймворк радить повторно розгорнути попередній артефакт, а не лагодити поточну інфраструктуру. Попередній образ — це відомий стан, який уже працював у продакшні, тож команді не доводиться під тиском відтворювати з пам’яті та історії оболонки, яким сервер був учора.

Третя причина — менше рук у продакшні. Фреймворк зазначає, що підключення до продакшн-серверів, щоб оновити їх вручну, породжує людські помилки на кшталт одруківок і пропущених кроків, а незмінна інфраструктура прибирає потребу в ручному доступі до серверів. Менше людей узагалі потребують входу на живу машину, а ті, хто його має, рідко ним користуються.

Четверта причина — слід для аудиту. Аудитори, як зазначено на тій самій сторінці, хочуть доказів, що продакшн не змінювали поза схваленими процесами, а ручні зміни лишають неповні записи. Коли кожна зміна —

новий артефакт, система контролю версій і журнали CI/CD показують, що змінилося, коли і хто це схвалив. Латка безпеки проходить тим самим шляхом, що й нова функція: вона стає новим артефактом, який замінює робочий, тож її збирають, тестують і фіксують так само.

Усе це має свою ціну. Фреймворк зауважує, що образи віртуальних машин потребують більше ресурсів, ніж контейнери, і довше запускаються, а контейнерам для роботи потрібен оркестратор. Кожна зміна, навіть однорядкове налаштування, тепер означає повну збірку й заміну, тож збірка має бути автоматичною і досить швидкою, щоб нікого не тягнуло її пропустити. Стан треба винести з машин, які замінюються, до першої заміни, а не після неї.

Як це виглядає на практиці

Шаблон Packer описує, як зібрати машинний образ. Цей шаблон побудовано на прикладі з документації до збирача Amazon EBS у Packer, а блок `build` додано у формі, яку показує довідка про блок build:

```
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")
  }
}
```

Блок `source` називає збирач і базовий образ `source_ami`, кореневий том якого копіюють і налаштовують на запущеному екземплярі. Блок `build` вказує, які джерела збирати і як їх налаштовувати. Провізіонер `shell` виконує два скрипти всередині цієї запущеної машини, перш ніж її

перетворять на статичний артефакт, і цей артефакт — новий образ. Оголошення двох змінних тут пропущено. Одна команда запускає всі збірки в шаблоні й створює артефакти:

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

Розгортання — друга половина. Фреймворк HashiCorp ставить два інструменти поруч: образи збирають за допомогою Packer, розгортають за допомогою Terraform і повністю замінюють, коли їх треба оновити. Коли зміну не можна внести на місці, Terraform за замовчуванням, згідно з довідкою про lifecycle, спершу знищує наявний об’єкт, а потім створює заміну. Для оновлень у цьому процесі не потрібно входити на запущений сервер робочого середовища. Зміна в скриптах дає новий образ, образ дає нові машини, а старі прибирають.

Та сама дисципліна працює і на одному хості. Запис у портфоліо про контейнери на Podman і Quadlet тримає підготовку хоста й розгортання окремо. Root один раз готує шлях для розгортання, а далі випуски робить звичайний користувач без особливих прав, якому дозволено запустити рівно один скрипт і більше нічого. Випуск — це новий образ і перезапуск служби, без оболонки root і без довільних команд. Запис про CI/CD на GitHub Actions іде далі з боку образів: фронтенд у продакшні працює на distroless-основі, де немає ні оболонки, ні пакетного менеджера, а збірку з одного середовища в наступне переводять робочі процеси (workflows), а не людина. У такому образі майже нема куди входити й нема чим щось установити, тож розумний спосіб його змінити — зібрати йому наступника.

Запис про сервери, які можна відтворити з нуля ретельно показує іншу модель. Ansible доводить хост, що вже працює, до стану, описаного в репозиторії, кожен play ідемпотентний, а репозиторій містить конфігурацію для відтворення описаного середовища. Запис не приховує, чого коштують зміни на місці. Будь-що змінювати довше, ніж просто поправити файл прямо на сервері, а прогін, що зупинився на півдорозі, лишає хост у гіршому стані, ніж прогін, який відмовився починати. Тому тепер перед ним виконується попередній плей (preflight). Хост, змінений наполовину, — це крок до того крихкого сервера, який описував Фаулер, а заміна машини замість її зміни — другий спосіб цей ризик оминути.

Де виникають проблеми

Модель легко сформулювати і легко розмити. Більшість збоїв починається з одного винятку, який на той момент здавався нешкідливим.

Швидке виправлення через SSH повертає дрейф. Під тиском хтось заходить на робочий сервер і редагує файл. Виправлення працює до наступного розгортання, яке замінює машину, а тоді зникає разом із будь-яким поясненням, навіщо воно було потрібне. А доти робочий сервер уже розходиться зі своїм образом, і це саме той стан, якому підхід мав запобігти. Вихід — проводити кожну зміну, навіть термінову, через збірку образу й тримати цю збірку настільки швидкою, щоб вона була найкоротшим шляхом. Настанова HashiCorp — ніколи не підключатися до робочих серверів, щоб вносити зміни вручну чи запускати команди оновлення.

Дані, записані на машину, видаляються разом із нею. Документація Docker каже, що коли контейнер видаляють, зміни у файлах, зроблені всередині нього, теж зникають. Завантаження, згенеровані файли чи локальна база даних на машині, яку замінюють, зникнуть із наступним релізом. Вихід — тримати стан поза тією частиною, яку замінюють. Томи Docker зберігають дані довше, ніж живе окремий контейнер, і новий контейнер, підключений до того самого тому, знову знаходить файли; для машин ту саму роль виконує окремий диск або сервіс бази даних поза образом.

Заміна зупиняє сервіс. Terraform за замовчуванням спершу знищує наявний об’єкт, а вже потім створює заміну, і так з’являється проміжок, коли сервіс узагалі не працює. Параметр `create_before_destroy` у блоці `lifecycle` ресурсу змінює порядок на протилежний:

```
resource "aws_instance" "web" {
  # ...
  lifecycle {
    create_before_destroy = true
  }
}
```

HashiCorp робить це опцією, яку треба ввімкнути свідомо, бо багато типів об’єктів мають вимоги до унікальних назв чи інші обмеження, які порушили б дві копії одночасно, а довідка просить розібратися з обмеженнями кожного типу ресурсу, перш ніж на це покладатися.

Нова збірка непомітно стартує з новішої бази. Замість фіксованого `source_ami` збирач Amazon EBS може взяти його з `source_ami_filter`, а коли `most_recent` має значення true, фільтр обирає найновіший образ, що підходить під умови; документація називає це найкориснішим для щоденної збірки дистрибутива. Це зручно, щоб не відставати, але тоді дві збірки того самого шаблону в різні дні починаються з різних баз, і повторна збірка старого релізу вже не відтворює його. Вихід для релізних образів — закріпити `source_ami` за конкретним образом і просувати його свідомо, окремим комітом, що пройшов рев'ю.

Секрети вшиті в образ. Фреймворк HashiCorp однозначний: артефакти ніколи не повинні містити секретів, як-от паролів до баз даних, ключів API чи ключів шифрування, бо будь-хто з доступом до артефакту може їх витягнути, а ротація облікових даних вимагає перезібрати й перерозгорнути кожен артефакт. Вихід — отримувати секрети під час запуску машини чи контейнера. Власний підхід HashiCorp спирається на Vault: машина, що стартує, проходить автентифікацію й отримує тимчасові облікові дані з обмеженим терміном дії, а облікові дані змінюють у Vault без жодної перезбірки. Шаблон вище дотримується того самого правила для власних ключів: вони надходять як змінні, а не записані текстом у файлі.

Попереднього артефакту немає саме тоді, коли він потрібен. Відкат означає повторне розгортання попереднього артефакту, а це передбачає, що він досі існує. Завдання очищення, яке видаляє старі образи, або звичка давати кожній збірці ту саму назву образу перетворюють відкат на перезбірку зі старого коду на базі, яка могла зрушити. Вихід — зберігати образи останніх релізів, що працювали в продакшні, давати кожному назву, яку ніколи не використовують повторно, і відрепетирувати відкат на один із них заздалегідь, до дня, коли це справді знадобиться.

Що почитати далі

- Create immutable infrastructure — Well-Architected Framework про дрейф, відкат, види образів і секрети.
- Trash Your Servers and Burn Your Code: Immutable Infrastructure and Disposable Components — допис Чеда Фаулера з червня 2013 року, прочитаний у копії Internet Archive.
- Introduction to Packer — що збирає Packer і що містить машинний образ.
- Amazon EBS builder — збирач із прикладу вище, з `source_ami` і `source_ami_filter`.

- lifecycle meta-argument reference — порядок заміни Terraform за замовчуванням і `create_before_destroy` .
- Persisting container data — чому файли в контейнері зникають разом із ним і як їх зберігають томи.

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 незмінна інфраструктура?» 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/immutable-infrastructure/