Engineer

Docker

способ постачати програму разом з її оточенням

Що це таке

Docker — це набір інструментів, який пакує програму разом із бібліотеками, середовищем мови та потрібними файлами в один артефакт, образ, а потім запускає цей образ як ізольований процес, тобто контейнер. Рецепт образу записують у Dockerfile. Зібраний один раз образ однаково стартує на ноутбуці розробника, на сервері збирання і в продакшні. У цьому вся обіцянка: працює саме те, що тестували, до останнього байта.

Словник тримається на чотирьох поняттях. Dockerfile — текстовий файл з інструкціями на кшталт `FROM`, `COPY` і `RUN`. Образ — шаблон лише для читання, зібраний за цими інструкціями і збережений як стос шарів, по одному на інструкцію, тож після дрібної правки перезбираються тільки ті шари, що змінилися. Контейнер — запущений екземпляр образу з тонким шаром для запису згори. Реєстр, як-от Docker Hub або реєстр хмарного провайдера, — місце, куди образи завантажують і звідки їх забирають.

За командним рядком стоять клієнт і сервер. Команда `docker` — це клієнт: вона надсилає запити через REST API, зазвичай через Unix-сокет, до демона `dockerd`, який працює постійно й володіє образами, контейнерами, мережами та томами. Сам життєвий цикл контейнера демон передає containerd, а той за замовчуванням використовує runc, невелику програму, що просить ядро Linux про простори імен і контрольні групи та запускає процес усередині них. Отже, контейнер — звичайний процес на ядрі хоста з обмеженим поглядом на систему, а не віртуальна машина. Саме тому Docker Desktop на macOS і Windows тримає у фоні невелику віртуальну машину з Linux: контейнерам потрібне ядро Linux, яке можна розділити.

Формат давно переріс компанію, що його створила. У червні 2015 року Docker, CoreOS та інші заснували Open Container Initiative, і Docker передав туди свій формат образів і runc. Специфікації, що з цього виросли, пояснюють, чому образ, зібраний у Docker, без змін запускається в Podman, containerd чи керованому хмарному сервісі.

Застосуйте поняття до рішення

Застосуйте «Що таке Docker?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Запакована так, що поводиться однаково на ноутбуці, на тестовому сервері й у продакшні, і масштабується, коли настає напружений тиждень.

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. Вела 20 застосунків Docker у Google Cloud і AWS — 20 контейнеризованих застосунків розгорнуто й поставлено під моніторинг

Вела 20 застосунків Docker у Google Cloud і AWS Зв'язатися з нами →

Чому це важливо

Питання «Docker чи ні» загалом мало що дає. Корисніше питати, чи заслуговує саме цей застосунок на образ, чи звичайна служба просто на хості зробить те саме з меншою кількістю рухомих частин.

Образ виправдовує себе, коли програма тягне за собою ціле оточення. Застосунок на Python чи R, якому потрібен точний набір пакетів, стара служба, прив’язана до давньої версії середовища, дві програми, що хочуть різних версій однієї бібліотеки: усе це важко відтворити вручну і легко відтворити з образу. Відкат означає запуск попереднього образу, а новий розробник виконує одну команду замість цілої сторінки інструкцій із налаштування. Порівняння Docker і віртуальної машини — той самий компроміс з іншого боку. Контейнери стартують за мить і ділять одне ядро, тож на тому самому хості їх уміщається в рази більше, але ізоляція слабша.

Ціна — справжня інфраструктура. За замовчуванням демон працює від root і керує мережею; Docker також підтримує режим rootless. Реєстр треба або тримати самому, або оплачувати. Базові образи накопичують виправлення безпеки, і щоб їх отримати, образ доводиться перезбирати. Сховище живе за іншими правилами, ніж файлова система хоста. Статичний бінарний файл без залежностей майже нічого від цього не виграє, і модуль systemd із пісочницею запустить його не гірше.

Сюди ж належить різниця між Docker і Podman. Обидва запускають ті самі образи й розуміють більшість тих самих команд, але Docker використовує демон, за замовчуванням привілейований, а Podman запускає контейнери напряму, як дочірні процеси команди або systemd. На ноутбуці різниця мала. На сервері, де хтось уже продумав брандмауер і нагляд за процесами, вона може вирішити справу.

Як це виглядає на практиці

Docker трапляється в роботах на цьому сайті часто, а один із проєктів — це свідоме рішення його не брати.

З 2022 по 2024 рік двадцять контейнеризованих застосунків працювали в Google Cloud і AWS, серед них аналітичні застосунки на R, де найважче щоразу отримати той самий набір пакетів. Ці двадцять застосунків розгортали й підтримували з узгодженою конфігурацією та моніторингом, завдяки чому вони лишалися стабільними й відтворюваними, а проблеми в обох хмарах діагностувалися швидше.

Сто Docker-застосунків працювали на хостах з Ubuntu за проксі на Apache і Nginx та потребували дійсних TLS-сертифікатів. Робочий процес на основі ACME випускав, поновлював і розгортав ці сертифікати автоматично, тож усі сто зберігали дійсні сертифікати, а ручна робота із сертифікатами зникла.

Третій приклад веде в протилежний бік. На хості, правила брандмауера якого задає роль посилення захисту, контейнерне середовище побудували на Podman і Quadlet замість Docker, бо стандартна мережа Docker переписує правила фільтра пакетів ядра й публікує порти контейнерів вище за власні правила брандмауера. Причину записано поруч із рішенням, тож наступна людина на цьому хості побачить, що стандартний варіант зважили, а не пропустили.

Де виникають проблеми

Перший контейнер піднімається легко. Клопіт приходить пізніше, і здебільшого в одній із чотирьох форм.

Порт, який закрив брандмауер, відповідає з інтернету. За замовчуванням опублікований порт прив'язується до всіх адрес хоста, а правила фільтра пакетів, які пише Docker, переспрямовують трафік до контейнера ще до того, як його побачить інструмент на кшталт ufw. Брандмауер показує порт закритим, а він відкритий. Якщо до контейнера має діставатися лише локальний зворотний проксі, публікуйте порт на адресі loopback:

```
# доступний ззовні, хоч би що показував ufw
docker run -d -p 8080:80 nginx:1.27

# доступний лише з самого хоста
docker run -d -p 127.0.0.1:8080:80 nginx:1.27
```

Дані зникають разом із контейнером. Усе, що записано у власний шар контейнера, видаляється разом із контейнером, а саме це й робить оновлення. Бази даних, завантажені файли та все інше, що має пережити оновлення, кладуть на іменований том або bind mount, і тоді вже цей том потребує власної резервної копії.

Нове збирання дає інший образ, ніж минулого тижня. Тег на кшталт `python:3` чи `latest` — рухомий вказівник, тож той самий Dockerfile іншого дня збирає інший образ. Закріплюйте базові образи за конкретною версією, а коли збирання має повторюватися точно, то за дайджестом, і перезбирайте свідомо, коли для базового образу виходять виправлення безпеки.

Облікові дані їдуть усередині образу. Аргументи збирання та змінні оточення лишаються у фінальному образі, а файл, скопійований в один шар, залишається в ньому, навіть якщо наступна інструкція його видалить. Прочитати його може кожен, хто здатен завантажити образ. Секрети слід передавати збиранню через secret mount: він відкриває їх лише одному кроку `RUN` і не записує в жоден шар.

Що почитати далі

- Docker Docs — офіційні посібники та довідка для рушія, командного рядка і Compose.
- Docker overview — клієнт, демон, образи й контейнери в поясненні самого проєкту.

- Port publishing and mapping — чому опубліковані порти за замовчуванням відкриті назовні і як прив’язати їх до localhost.
- Packet filtering and firewalls — як Docker пише правила iptables або nftables і де ufw перестає бачити трафік.
- Build secrets — як передати збиранню облікові дані, не залишивши їх в образі.
- OCI Image Format Specification — відкритий стандарт, завдяки якому один образ працює в Docker, Podman і containerd.

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 Docker?» 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/docker/