Engineer

Перевірка справності

автоматичний запит, який показує, чи сервіс працює

Що це таке

Перевірка справності (health check) — це невеликий тест, який повторюється за розкладом і запитує працюючий сервіс, чи може він виконувати свою роботу, разом із правилом, що робити, коли відповідь негативна. Тестом може бути HTTP-запит до окремого шляху, команда, запущена всередині контейнера, або спроба відкрити TCP-порт. Саме правило перетворює перевірку з вимірювання на механізм керування: невдала перевірка може вивести сервер із ротації, перезапустити контейнер, зупинити розгортання або розбудити когось посеред ночі.

Docker вбудовує цю ідею просто в образ. Інструкція `HEALTHCHECK` у Dockerfile задає команду, яку Docker запускає всередині контейнера з фіксованим інтервалом. Код виходу 0 означає, що контейнер справний, 1 — що несправний, а код 2 зарезервовано, і використовувати його не можна. Контейнер із перевіркою починає в стані `starting`, переходить у `healthy`, щойно одна перевірка пройде, і стає `unhealthy` після заданої кількості невдач поспіль. Типові значення — перевірка кожні 30 секунд, тайм-аут 30 секунд і три спроби; запуск, що перевищив тайм-аут, зараховується як невдача, а його процес примусово завершується. Необов'язковий стартовий період дає повільному застосунку час запуститися, і невдачі в цей період не зараховуються. У Dockerfile може бути лише одна така інструкція: якщо їх кілька, діє остання, а `HEALTHCHECK NONE` вимикає перевірку, успадковану від базового образу.

Kubernetes розбиває це питання на три проби, і кожна має власний наслідок. Startup-проба з'ясовує, чи застосунок завершив запуск; доки вона не пройде, дві інші чекають, а якщо вона раз у раз провалюється, контейнер завершують, і далі з ним чинять згідно з його політикою перезапуску. Liveness-проба з'ясовує, чи треба перезапустити контейнер, і призначена для станів на кшталт взаємного блокування (deadlock), коли процес працює, але

не просувається вперед. Readiness-проба з'ясовує, чи має контейнер отримувати трафік. Коли вона провалюється, контейнер не перезапускають; натомість адресу Pod вилучають з ендпоінтів кожного Service, чий селектор його обирає, і запити йдуть до інших реплік, доки проба знову не пройде.

Кожна проба використовує один із чотирьох механізмів. Проба `httpGet` проходить за будь-якого коду статусу від 200 до 399 включно. Проба `tcpSocket` проходить, якщо порт приймає з'єднання, навіть таке, що одразу закривається. Проба `grpc` проходить, коли сервіс відповідає `SERVING` за протоколом перевірки справності gRPC. Проба `exec` виконує команду в контейнері й проходить, якщо та завершилася з кодом 0. Якщо не вказано інше, Kubernetes перевіряє кожні десять секунд, чекає на відповідь одну секунду й реагує після трьох невдач поспіль.

Балансувальники навантаження додають ще один різновид. Описані вище перевірки активні: щось надсилає тестовий запит незалежно від того, чи є зараз користувачі. Пасивна перевірка натомість стежить за справжнім трафіком. Саме так працює nginx з відкритим кодом: коли запит до upstream-сервера закінчується помилкою, nginx позначає цей сервер як несправний і на якийсь час його оминає, а за типового значення `max_fails`, що дорівнює 1, для цього досить однієї невдалої спроби. Документація nginx зараховує перевірки справності застосунків (application health checks) до можливостей платних підписок NGINX Plus. Пасивна перевірка не коштує додаткових ресурсів, але про несправний сервер вона дізнається з невдалого запиту користувача і мовчить про сервер, на який трафік не надходить.

Застосуйте поняття до рішення

Застосуйте «Що таке перевірка справності?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Моніторинг і сповіщення, які попереджають, поки збій ще малий, і цифри, що показують, як часто так і залишається.

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. Запустила перший дашборд спостережуваності — перший огляд власних систем компанії

Запустила перший дашборд спостережуваності Зв'язатися з нами →

Чому це важливо

Працюючий процес — ще не справний сервіс. Довідник Docker наводить типовий приклад: вебсервер, що застряг у нескінченному циклі й перестав приймати з'єднання, хоча його процес досі живий. Усе, що стежить лише за

тим, чи існує процес, як-от юніт systemd із політикою перезапуску, не бачить жодної проблеми. Розрізнити ці стани може тільки питання, поставлене через ті самі двері, якими заходять користувачі, і в цьому весь сенс перевірки справності: вона замінює питання «чи він запущений?» питанням «чи він відповідає?».

Два рішення визначають, чи буде перевірка корисною, чи небезпечною. Перше — що саме вона перевіряє, а це, по суті, визначення слова «працює», записане в коді. Обробник, який повертає 200, нічого не торкаючись, доводить лише, що вебфреймворк запущено; обробник, який ще й звертається до бази даних, доводить, що база досяжна, і провалюється щоразу, коли вона недосяжна. Друге — що відбувається в разі невдачі, і від цього залежить радіус ураження. Вивести одну репліку з ротації дешево, і це легко скасувати. Перезапуск викидає її незавершену роботу й прогріті кеші, а якщо всі репліки провалюють ту саму перевірку одночасно, перезапуск усіх перетворює повільну залежність на повну відмову сервісу.

Kubernetes попереджає саме про це. Документація каже, що liveness-проби треба налаштовувати так, щоб вони справді вказували на збій, з якого застосунок не відновиться сам, і що неправильна реалізація може спричинити каскадні збої: перезапуски контейнерів під високим навантаженням, невдалі клієнтські запити й додаткову роботу для подів, що лишилися. Звідси порада тримати ці два питання окремо. Коли застосунок жорстко залежить від бекенд-сервісів, його liveness-проба має проходити, доки справний сам застосунок, а readiness-проба має перевіряти ще й кожен потрібний бекенд. Тоді трафік оминає Pod, який міг би відповідати лише помилками, а сам Pod не вбивають через несправність, що криється деінде.

Перевірки справності також працюють на моніторинг. Ендпоінт, який опитує балансувальник навантаження, можна опитувати й ззовні, і тривога, що спрацьовує, коли він падає, часто стає першою ознакою інциденту. Коли Irakli Nadareishvili писав Internet-Draft Health Check Response Format for HTTP APIs, стандарту на те, що має повертати такий ендпоінт, не існувало, і більшість застосунків покладалися або лише на код статусу HTTP, або на власний формат. Чернетка пропонувала JSON-відповідь із типом медіа `application/health+json` і обов'язковим полем `status` зі значенням `pass`, `fail` або `warn`, де `pass` і `warn` поєднуються з кодом статусу 2xx чи 3xx, а `fail` — з кодом 4xx чи 5xx. Остання редакція, від жовтня 2021 року, втратила чинність у квітні 2022-го, і IETF зазначає, що IETF її не схвалювала і формального статусу вона не має. Вона корисна як спільний словник, а не як стандарт.

Як це виглядає на практиці

У Dockerfile перевірка постачається разом з образом, який вона перевіряє. Тут власна невелика програма-проба застосунку запитує шлях на localhost і завершується з кодом 0 або 1:

```
HEALTHCHECK --interval=15s --timeout=3s --start-period=60s --retries=3 \
  CMD ["/app/bin/probe", "http://127.0.0.1:8080/livez"]
```

Docker запускає пробу кожні 15 секунд і дає кожному запуску три секунди. Невдачі в першу хвилину не зараховуються, доки хоч одна перевірка не пройде успішно, а в Docker Engine 25.0 і новіших перевірки в цю хвилину виконуються кожні п’ять секунд — це типовий стартовий інтервал, — тож швидкий запуск помічають швидко. Після трьох невдач поспіль контейнер позначається як `unhealthy`. Перші 4096 байтів виводу проби зберігаються для `docker inspect`, а кожна зміна стану породжує подію `health_status`. Довідник описує стан, а не дію: реагувати на нього має щось інше.

У Kubernetes трьом пробам можна задати різні шляхи, щоб кожна ставила своє питання:

```
containers:
  - name: api
    ports:
      - containerPort: 8080
    startupProbe:
      httpGet: { path: /livez, port: 8080 }
      periodSeconds: 5
      failureThreshold: 24 # до 2 хвилин на запуск
    livenessProbe:
      httpGet: { path: /livez, port: 8080 }
      periodSeconds: 10
      timeoutSeconds: 2
      failureThreshold: 6 # близько хвилини до перезапуску
    readinessProbe:
      httpGet: { path: /readyz, port: 8080 }
      periodSeconds: 5
      timeoutSeconds: 2
      failureThreshold: 2
```

`/livez` відповідає лише за сам процес, а `/readyz` перевіряє ще й базу даних та все інше, без чого запит не обійдеться. Startup-проба використовує шлях liveness, як радить документація Kubernetes, і дає 24 × 5 секунд, перш ніж контейнер вважатимуть безнадійним. Щойно вона пройде, керування переходить до liveness із довгим запалом і до readiness із коротким: близько десяти секунд невдач виводять Pod із Service, а хвилина невдач перезапускає його. Типові значення дали б три невдачі з інтервалом у десять секунд, а цього замало для застосунку, якому потрібно дві хвилини, щоб завантажити свої дані. Власне правило документації таке: контейнеру, якому на запуск зазвичай потрібно більше, ніж initialDelaySeconds плюс failureThreshold × periodSeconds, потрібна startup-проба.

Перевірку справності можна й вивернути навиворіт. У моніторингу хоста з dead man’s switch, зробленому у вересні 2026 року, хост із 464 МБ пам’яті, на якому працюють git-сервер і вебсервер, перевіряє сам себе кожні п’ятнадцять хвилин із випадковим зсувом: невеликий скрипт вимірює доступну пам’ять і вільне місце на диску й пінгує зовнішній сервіс, лише якщо обидва показники вищі за свої пороги. Тривогу здіймає тиша. Машина, у якої скінчилася пам’ять, пропала мережа або яка перестала завантажуватися, дає той самий сигнал, що й просто несправна, і саме тому обрали такий підхід, а не агента, який звітує про стан. Перевірка читає доступну пам’ять, а не вільну, бо ядро віддає незайняту пам’ять під кеш, і шість імен змінних зі словом «free» виправили, щоб ніхто не «полагодив» робочу сигналізацію, перетворивши її на таку, що постійно перебуває за порогом. Отриманий сигнал підтверджує, що скрипт виконався, а перевірки доступної пам’яті й вільного місця на диску пройдено за встановленими порогами; він не засвідчує загальної справності хоста чи того, що сайт обслуговує запити. Моніторинг може виявляти тривожні ознаки, хоча час їх появи залежить від навантаження й умов спостереження.

Перевірка під час запуску відповідає на ще одне питання. У пошті з failover і журналом доставки 2025 року листи йдуть через трьох провайдерів з автоматичним резервним перемиканням, а на вході до всього цього стоїть стартова перевірка, яку вмикає прапорець: вона надсилає справжній лист через увесь стек. У середовищі розробки невдача лише записує попередження в журнал, і застосунок працює далі; у staging і production вона фатальна, і процес завершується, щоб не розгорнути збірку, яка не здатна надсилати пошту. Сторінка підсумовує результат так: цілий клас тихих збоїв перейшов із розряду «користувач помітить через кілька днів» у розряд «розгортання зупиняється».

Де виникають проблеми

Перевірка доводить лише, що процес відповідає. Обробник, який повертає 200, нічого не торкаючись, лишається зеленим, коли база даних недосяжна, диск заповнений або кожен справжній запит закінчується помилкою. Нехай readiness-перевірка дешево випробовує те, що потрібно запиту, і додайте перевірку ззовні, яка проходить шляхом користувача через DNS, TLS і балансувальник навантаження, бо kubelet надсилає проби прямо на власну IP-адресу Pod і нічого з цього не бачить.

Liveness залежить від чогось поза контейнером. Якщо liveness-проба звертається до бази даних, повільна база валить її в усіх репліках одночасно, і Kubernetes перезапускає їх усі саме тоді, коли сервісу й так важко. Нехай liveness відповідає лише за сам процес, як радить документація Kubernetes, а залежності перенесіть у readiness, де невдача відводить трафік, а не вбиває контейнери.

Часові параметри обрано без огляду на застосунок. За замовчуванням Kubernetes чекає на відповідь одну секунду, тож ендпоінт, який сповільнюється під навантаженням, провалює перевірку, хоча застосунок просто зайнятий, а контейнер, який запускається повільніше, ніж дозволяє його liveness-проба, вбивають ще до того, як він буде готовий, і так раз за разом. Виміряйте, скільки часу ендпоінт відповідає під навантаженням, задайте тайм-аут із запасом понад це значення, а повільним застосункам дайте startup-пробу, розраховану на найповільніший реальний запуск.

Перенаправлення проходить. HTTP-проба в Kubernetes вважає успіхом будь-який статус від 200 до 399, тож шлях, що перенаправляє на сторінку входу, виглядає справним. Якщо перенаправлення веде на інший хост, kubelet за ним не переходить, а записує пробу як успішну з попередженням. Зі схемою HTTPS kubelet до того ж пропускає перевірку сертифіката, тож прострочений сертифікат ніколи не провалить пробу. Спрямовуйте проби на шлях, який відповідає 200 напряму й повертає 5xx, коли щось не так, а сертифікати перевіряйте ззовні.

Ендпоінт розповідає забагато й забагатьом. Детальна відповідь, що перелічує кожну залежність, версію та ім’я хоста, інформує зловмисника не гірше, ніж балансувальник навантаження, а розділ Internet-Draft про безпеку зазначає, що ендпоінтам перевірки справності може знадобитися

автентифікація й рольовий контроль доступу. Залиште в публічній відповіді лише статус, а подробиці віддавайте на внутрішньому порту або тільки автентифікованим клієнтам.

Перевірка коштує більше, ніж мала б. Kubernetes попереджає, що проби `exec` під час кожного запуску створюють нові процеси, і на щільно заповнених вузлах із короткими інтервалами це може додати навантаження на процесор вузла. Проба, яка кожні кілька секунд виконує важкий запит на кожній репліці, сама по собі стає навантаженням. Віддавайте перевагу пробам HTTP, TCP або gRPC, тримайте ендпоінт дешевим і кешуйте результат дорогої перевірки залежностей на кілька секунд, а не повторюйте її для кожної проби.

Що почитати далі

- Dockerfile reference: HEALTHCHECK — інструкція, її параметри й типові значення та що означає кожен код виходу.
- Liveness, Readiness, and Startup Probes — що робить кожен тип проби, чотири механізми перевірки та всі поля конфігурації з їхніми типовими значеннями.
- Configure Liveness, Readiness and Startup Probes — покрокові приклади, зокрема startup-проба, яка дає повільному застосунку п’ять хвилин.
- Health Check Response Format for HTTP APIs — чернетка Internet-Draft, що вже втратила чинність, на якій ґрунтується `application/health+json`, та її статуси pass, fail і warn.
- Using nginx as HTTP load balancer — пасивні вбудовані (in-band) перевірки справності в nginx з відкритим кодом і параметри `max_fails` та `fail_timeout`.

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 перевірка справності?» 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/health-check/