Engineer

Comprobación de salud

Prueba automática para comprobar si un servicio funciona

Qué es

Una comprobación de salud es una pequeña prueba, repetida según un calendario, que pregunta a un servicio en ejecución si puede cumplir su función, junto con una regla que determina qué ocurre si la respuesta es negativa. La prueba puede ser una solicitud HTTP a una ruta dedicada, un comando dentro de un contenedor o un intento de abrir un puerto TCP. La regla la convierte de medición en control: una comprobación fallida puede retirar un servidor de la rotación, reiniciar un contenedor, detener un despliegue o despertar a alguien.

Docker incorpora la idea a la imagen. La instrucción `HEALTHCHECK` de un Dockerfile indica un comando que Docker ejecuta dentro del contenedor a intervalos fijos. El estado de salida 0 significa sano; el 1, no sano; el 2 está reservado y no debe usarse. Un contenedor con comprobación empieza en `starting`, pasa a `healthy` en cuanto supera una y se vuelve `unhealthy` tras un número definido de fallos consecutivos. Los valores predeterminados son una comprobación cada 30 segundos, un timeout de 30 segundos y tres reintentos; una ejecución que supera su timeout cuenta como fallo y se termina su proceso. Un período de inicio opcional da tiempo a arrancar a una aplicación lenta y no cuenta los fallos ocurridos dentro de él. Un Dockerfile contiene una instrucción de este tipo; si hay varias prevalece la última, y `HEALTHCHECK NONE` desactiva una comprobación heredada de la imagen base.

Kubernetes divide la pregunta en tres sondas, cada una con su consecuencia. La sonda de inicio pregunta si la aplicación ha terminado de arrancar; hasta que lo consigue no se ejecutan las otras dos y, si sigue fallando, se termina el contenedor y se aplica su política de reinicio. La sonda de vitalidad pregunta si hay que reiniciar el contenedor y está pensada para estados como un interbloqueo, donde el proceso se ejecuta pero no avanza. La sonda de preparación pregunta si el contenedor debe recibir tráfico. Si falla no se reinicia

nada: la dirección del Pod se retira de los endpoints de todos los Service que lo seleccionan y las solicitudes pasan a las otras réplicas hasta que la sonda vuelve a superarse.

Cada sonda usa uno de cuatro mecanismos. Una `httpGet` se supera con cualquier código de estado desde 200 hasta menos de 400. Una `tcpSocket` se supera si el puerto acepta una conexión, aunque se cierre de inmediato. Una `grpc` se supera cuando la comprobación de salud gRPC del servicio responde `SERVING`. Una `exec` ejecuta un comando en el contenedor y se supera con el estado de salida 0. Salvo que se indique lo contrario, Kubernetes comprueba cada 10 segundos, espera un segundo la respuesta y actúa tras tres fallos seguidos.

Los balanceadores de carga añaden un segundo tipo. Las comprobaciones anteriores son activas: algo envía una solicitud de prueba haya o no usuarios. Una comprobación pasiva observa el tráfico real. Así funciona nginx de código abierto: cuando una solicitud a un servidor de backend (upstream) termina en error, nginx lo marca como fallido y lo evita durante un tiempo; con el `max_fails` predeterminado de 1, basta un único intento fallido. La documentación de nginx incluye las comprobaciones de salud de aplicaciones entre las funciones de las suscripciones de pago NGINX Plus. Una comprobación pasiva no tiene coste adicional, pero descubre un servidor averiado a partir de una solicitud fallida de un usuario y no dice nada de un servidor que no recibe tráfico.

Utilice el concepto en una decisión

Aplique «¿Qué es una comprobación de salud?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Monitorización y alertas que avisan cuando un fallo aún es pequeño, y cifras que muestran con qué frecuencia se mantiene así.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Primer panel de observabilidad de la empresa — la primera visión de la empresa sobre sus propios sistemas

Primer panel de observabilidad de la empresa Póngase en contacto →

Por qué importa

Un proceso en ejecución no equivale a un servicio que funciona. La referencia de Docker da el caso típico: un servidor web atrapado en un bucle infinito que ya no acepta conexiones, aunque su proceso siga vivo. Todo lo que solo vigila si existe el proceso, como una unidad de systemd con política de reinicio, no ve ningún

problema. Solo una pregunta por la misma puerta que usan los usuarios permite distinguirlo; esa es toda la razón de la comprobación de salud: sustituye «¿está en ejecución?» por «¿responde?».

Dos decisiones hacen que una comprobación resulte útil o peligrosa. La primera es qué comprueba, lo que equivale a definir «funcionar» mediante código. Un manejador que devuelve 200 sin tocar nada prueba que el framework web está activo; otro que también consulta la base de datos prueba además que es accesible y falla siempre que no lo sea. La segunda es qué ocurre al fallar, lo que fija el alcance del daño. Retirar una réplica de la rotación es barato y reversible. Reiniciarla descarta el trabajo en curso y las cachés ya cargadas; si todas las réplicas fallan la misma comprobación a la vez, reiniciarlas convierte una dependencia lenta en una caída completa.

Kubernetes advierte precisamente de esto. Su documentación dice que las sondas de vitalidad deben configurarse para indicar de verdad un fallo del que la aplicación no puede recuperarse por sí sola y que una implementación incorrecta puede causar fallos en cascada: contenedores reiniciados con carga alta, solicitudes de clientes fallidas y más trabajo sobre los pods restantes. El consejo es separar las dos preguntas. Si una aplicación depende estrictamente de servicios de backend, la sonda de vitalidad debería superarse mientras la propia aplicación esté sana, y la de preparación debería comprobar también cada backend necesario. El tráfico evita así un Pod que solo podría responder con errores y el Pod no se termina por un fallo situado en otra parte.

Las comprobaciones de salud también sirven a la monitorización. Un endpoint que sondea un balanceador puede sondearse desde fuera, y la alerta que salta cuando falla suele ser la primera señal de un incidente. Cuando Irakli Nadareishvili escribió el borrador de Internet «Formato de respuesta de las comprobaciones de salud para API HTTP», no había un estándar para la respuesta de estos endpoints y la mayoría de las aplicaciones dependían solo del estado HTTP o de un formato propio. El borrador propuso un cuerpo JSON con el tipo de medio `application/health+json` y un campo obligatorio `status` con `pass`, `fail` o `warn`: `pass` y `warn` acompañan a un estado 2xx o 3xx, y `fail` a uno 4xx o 5xx. Su última revisión, de octubre de 2021, caducó en abril de 2022; la IETF lo presenta sin respaldo y sin reconocimiento formal. Es útil como vocabulario compartido, no como estándar.

Cómo se aplica en la práctica

En un Dockerfile, la comprobación se distribuye con la imagen que comprueba. Aquí, el pequeño programa de sondeo de la aplicación solicita una ruta en localhost y termina con 0 o 1:

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

Docker ejecuta la sonda cada 15 segundos y concede tres segundos a cada ejecución. Los fallos del primer minuto no cuentan salvo que ya se haya superado una comprobación; en Docker Engine 25.0 o posterior, las comprobaciones de ese minuto se ejecutan cada cinco segundos, el intervalo de inicio predeterminado, de modo que un arranque rápido se detecta pronto. Tras tres fallos consecutivos, el contenedor se marca como `unhealthy` . Los primeros 4.096 bytes de salida se conservan para `docker inspect` , y cada cambio de estado emite un evento `health_status` . La referencia describe un estado, no una acción: otra cosa debe actuar sobre él.

En Kubernetes, las tres sondas pueden tener rutas distintas para que cada una haga su propia pregunta:

```
containers:
  - name: api
    ports:
      - containerPort: 8080
    startupProbe:
      httpGet: { path: /livez, port: 8080 }
      periodSeconds: 5
      failureThreshold: 24 # up to 2 minutes to start
    livenessProbe:
      httpGet: { path: /livez, port: 8080 }
      periodSeconds: 10
      timeoutSeconds: 2
      failureThreshold: 6 # about a minute before a restart
    readinessProbe:
      httpGet: { path: /readyz, port: 8080 }
      periodSeconds: 5
      timeoutSeconds: 2
      failureThreshold: 2
```

`/livez` responde solo por el proceso, y `/readyz` comprueba también la base de datos y todo lo que una solicitud necesita. La sonda de inicio usa la ruta de vitalidad, como recomienda la documentación de Kubernetes, y permite 24 × 5 segundos antes de dar por perdido el contenedor. Una vez superada, la vitalidad toma el relevo con un margen largo y la preparación con uno corto: unos diez segundos de fallos retiran el Pod del Service y un minuto de ellos lo reinicia. Los valores predeterminados permitirían tres fallos separados por diez segundos, demasiado poco para una aplicación que necesita dos minutos para cargar sus datos. La regla de la propia documentación es que un contenedor que suele tardar más en arrancar que la demora inicial más failureThreshold × periodSeconds necesita una sonda de inicio.

Una comprobación de salud también puede invertirse. En la monitorización del host mediante un interruptor de hombre muerto, de septiembre de 2026, un host de 464 MB que ejecuta una forja y un servidor web se comprueba cada quince minutos, con una variación aleatoria: un pequeño script mide la memoria disponible y el espacio libre en disco y solo envía una señal a un servicio externo si ambos superan sus umbrales. El silencio es la alerta. Un host sin memoria, sin red o que deja de arrancar envía la misma señal que otro simplemente no sano; por eso se eligió esta forma frente a un agente que informa de un estado. Se lee memoria disponible en vez de libre porque el núcleo llena la memoria ociosa con caché, y se corrigieron seis nombres de variables que contenían «free» («libre») para que nadie «arreglara» una alarma correcta convirtiéndola en otra que se dispara permanentemente. Una señal recibida confirma que el script se ejecutó y se superaron sus comprobaciones de los umbrales de memoria disponible y espacio libre en disco; no certifica el buen estado general del host ni que el sitio esté sirviendo páginas. La monitorización puede revelar señales de advertencia, aunque el momento en que aparecen depende de la carga de trabajo y de las condiciones observadas.

Una comprobación al arrancar responde a otra pregunta. En el correo con conmutación por error y registro de entregas, trabajo realizado en un puesto desempeñado desde 2025, el correo pasa por tres proveedores con conmutación automática y, por delante, una comprobación de inicio controlada por una bandera envía un mensaje real por toda la pila. En desarrollo, un fallo registra una advertencia y la aplicación continúa; en preproducción y producción es fatal, y el proceso termina en vez de desplegar una compilación que no puede enviar correo. La página resume el resultado como toda una categoría de fallos silenciosos que pasa de «un usuario lo descubre días después» a «el despliegue se detiene».

Dónde surgen los problemas

La comprobación solo demuestra que el proceso responde. Un manejador que devuelve 200 sin tocar nada sigue en verde aunque la base de datos sea inaccesible, el disco esté lleno o todas las solicitudes reales terminen en error. Haga que la comprobación de preparación ejercite lo que necesita una solicitud, con poco coste, y añada una comprobación externa que siga el recorrido de un usuario por DNS, TLS y el balanceador, ya que el kubelet envía sus sondas directamente a la IP del Pod y no ve nada de eso.

La vitalidad depende de algo externo al contenedor. Si su sonda consulta la base de datos, una base de datos lenta la hace fallar en todas las réplicas a la vez y Kubernetes las reinicia cuando el servicio ya tiene dificultades. Deje que la vitalidad responda solo por el proceso, como aconseja Kubernetes, y sitúe las dependencias en la preparación, donde un fallo desvía el tráfico en lugar de terminar contenedores.

Los tiempos no encajan con la aplicación. Kubernetes espera de forma predeterminada un segundo la respuesta, así que un endpoint que se ralentiza bajo carga falla cuando la aplicación solo está ocupada; un contenedor que arranca más despacio de lo permitido por su sonda de vitalidad se termina antes de estar listo, una y otra vez. Mida cuánto tarda el endpoint bajo carga y fije un timeout superior, y dé a los arranques lentos una sonda de inicio dimensionada para el arranque real más lento.

Una redirección supera la prueba. Una sonda HTTP de Kubernetes cuenta como éxito cualquier estado de 200 a 399, por lo que una ruta que redirige a un inicio de sesión parece sana. Si redirige a otro host, el kubelet no la sigue y registra la sonda como exitosa, con una advertencia. Al usar HTTPS también omite la verificación del certificado, así que uno caducado nunca hace fallar una sonda. Dirija las sondas a una ruta que responda 200 directamente y devuelva un 5xx si algo falla, y compruebe los certificados desde fuera.

El endpoint de salud revela demasiado a demasiadas personas. Una respuesta detallada que enumera cada dependencia, versión y nombre de host informa tanto a un atacante como a un balanceador; las consideraciones de seguridad del borrador de Internet señalan que estos endpoints pueden necesitar autenticación y control de acceso basado en roles. Limite la respuesta pública a un estado y sirva los detalles en un puerto interno o solo a clientes autenticados.

La comprobación cuesta más de lo debido. Kubernetes advierte que las sondas `exec` crean procesos en cada ejecución, lo que puede añadir consumo de CPU en nodos densamente ocupados y con intervalos cortos; una sonda que ejecuta una consulta pesada cada pocos segundos en cada réplica constituye carga por sí misma. Prefiera sondas HTTP, TCP o gRPC, procure que el endpoint consuma pocos recursos y almacene unos segundos en caché el resultado de una comprobación costosa de una dependencia en vez de repetirla en cada sondeo.

Lecturas adicionales

- Referencia de Dockerfile: HEALTHCHECK — La instrucción, sus opciones y valores predeterminados, y el significado de cada estado de salida.
- Sondas de vitalidad, preparación e inicio — Qué hace cada tipo, los cuatro mecanismos y todos los campos de configuración con sus valores predeterminados.
- Configurar sondas de vitalidad, preparación e inicio — Ejemplos desarrollados, incluida una sonda de inicio que concede cinco minutos a una aplicación lenta.
- Formato de respuesta de las comprobaciones de salud para API HTTP — El borrador de Internet caducado del que procede `application/health+json` y sus estados pass, fail y warn.
- Usar nginx como balanceador HTTP — Comprobaciones pasivas sobre tráfico real en nginx de código abierto y ajustes `max_fails` y `fail_timeout`.

Glosario

Elija según sus restricciones reales

Describa la ruta de despliegue, los riesgos operativos y quiénes se encargan de las guardias. Podemos definir el alcance de un cambio con comprobaciones, una vía de recuperación y responsabilidades claras.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es una comprobación de salud?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

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

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/health-check/