Engineer

Contrôle de santé

Test automatique du bon fonctionnement d'un service

Ce que c’est

Un contrôle de santé est un petit test répété à intervalles planifiés, qui demande à un service en cours d’exécution s’il peut remplir sa fonction, associé à une règle définissant ce qui se passe si la réponse est non. Le test peut être une requête HTTP vers un chemin dédié, une commande exécutée dans un conteneur ou une tentative de connexion à un port TCP. La règle transforme une mesure en mécanisme de contrôle : un échec peut retirer un serveur de la rotation, redémarrer un conteneur, arrêter un déploiement ou réveiller quelqu’un.

Docker inscrit l’idée dans l’image. L’instruction `HEALTHCHECK` d’un Dockerfile désigne une commande exécutée dans le conteneur à intervalle fixe. Le code de sortie 0 signifie sain, 1 défaillant ; 2 est réservé et ne doit pas être utilisé. Un conteneur contrôlé commence dans l’état `starting`, devient `healthy` dès qu’un contrôle réussit, puis `unhealthy` après un nombre défini d’échecs consécutifs. Sans autre configuration, le contrôle a lieu toutes les 30 secondes, avec un délai de 30 secondes et un seuil de trois échecs consécutifs ; un dépassement compte comme un échec et le processus du contrôle est tué. Une période de démarrage facultative laisse le temps à une application lente de démarrer, sans compter les échecs pendant cette période. Un Dockerfile contient une telle instruction ; s’il y en a plusieurs, la dernière l’emporte, et `HEALTHCHECK NONE` désactive un contrôle hérité de l’image de base.

Kubernetes divise la question en trois sondes, chacune avec sa conséquence. La sonde de démarrage demande si l’application a fini de démarrer ; tant qu’elle n’a pas réussi, les deux autres sont suspendues, et si elle continue d’échouer, le conteneur est tué puis traité selon sa politique de redémarrage. La sonde de vitalité demande s’il faut redémarrer le conteneur ; elle vise des états comme un interblocage, où le processus s’exécute sans progresser. La sonde de disponibilité demande si le conteneur doit recevoir du trafic. En cas d’échec, rien ne

redémarre : l'adresse du Pod est retirée des points de terminaison de chaque Service qui le sélectionne, et les requêtes vont vers les autres répliques jusqu'à ce que la sonde réussisse de nouveau.

Chaque sonde utilise l'un de quatre mécanismes. Une sonde `httpGet` réussit pour tout code de statut de 200 inclus à 400 exclu. Une sonde `tcpSocket` réussit si le port accepte une connexion, même fermée aussitôt. Une sonde `grpc` réussit lorsque le contrôle gRPC du service répond `SERVING`. Une sonde `exec` lance une commande dans le conteneur et réussit avec le code de sortie 0. Sauf indication contraire, Kubernetes sonde toutes les 10 secondes, attend une seconde la réponse et agit après trois échecs consécutifs.

Les répartiteurs de charge ajoutent un second type. Les contrôles précédents sont actifs : un composant envoie une requête de test, qu'il y ait ou non des utilisateurs. Un contrôle passif observe au contraire le trafic réel. C'est ainsi que fonctionne nginx open source : lorsqu'une requête vers un serveur en amont se termine en erreur, nginx le marque défaillant et l'évite quelque temps. Avec la valeur prédéfinie de 1 pour `max_fails`, une seule tentative ratée suffit. La documentation de nginx cite les contrôles de santé applicatifs parmi les fonctions des abonnements payants NGINX Plus. Un contrôle passif ne coûte rien de plus, mais apprend qu'un serveur est cassé par l'échec d'une requête utilisateur et ne dit rien d'un serveur sans trafic.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce qu'un contrôle de santé ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Une supervision et des alertes qui signalent les incidents tant qu'ils restent mineurs, avec les chiffres qui montrent à quelle fréquence ils le restent.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Premier tableau de bord d'observabilité — première vue de l'entreprise sur ses propres systèmes

Premier tableau de bord d'observabilité Contactez-nous →

Pourquoi c'est important

Un processus en cours d'exécution n'est pas un service qui fonctionne. La référence Docker donne le cas typique : un serveur web pris dans une boucle infinie qui n'accepte plus de connexions, alors que son processus tourne encore. Tout ce qui vérifie seulement l'existence du processus, comme une unité systemd avec politique de redémarrage, ne voit aucun problème. Seule une question

passant par la même porte que les utilisateurs distingue les deux ; c'est toute la raison d'être du contrôle de santé : remplacer « est-il en cours d'exécution ? » par « répond-il ? ».

Deux décisions rendent un contrôle utile ou dangereux. La première est ce qu'il teste : une définition du mot « fonctionner » écrite dans du code. Un gestionnaire qui renvoie 200 sans rien solliciter prouve que le framework web fonctionne ; s'il interroge aussi la base de données, il prouve qu'elle est accessible et échoue dès qu'elle ne l'est plus. La seconde est la conséquence de l'échec, qui fixe l'étendue des dégâts. Retirer une réplique de la rotation est peu coûteux et réversible. La redémarrer abandonne ses opérations en cours et ses caches chauds ; si toutes les répliques échouent au même contrôle en même temps, les redémarrer transforme une dépendance lente en panne totale.

Kubernetes met précisément en garde contre cela. Sa documentation indique que les sondes de vitalité doivent être configurées pour signaler réellement une défaillance irrécupérable, et qu'une mauvaise implémentation peut provoquer des pannes en cascade : conteneurs redémarrés sous forte charge, requêtes clientes en échec et davantage de travail sur les pods restants. Le conseil est de séparer les deux questions. Quand une application dépend strictement de services backend, sa sonde de vitalité doit réussir tant que l'application elle-même est saine, tandis que la sonde de disponibilité vérifie aussi chaque backend requis. Le trafic évite ainsi un Pod qui ne pourrait répondre que par des erreurs, sans le tuer pour une panne située ailleurs.

Les contrôles de santé servent aussi à la supervision. Un point de terminaison interrogé par un répartiteur peut l'être depuis l'extérieur, et l'alerte déclenchée par son échec est souvent le premier signe d'un incident. Quand Irakli Nadareishvili a rédigé le document de travail Internet-Draft intitulé Health Check Response Format for HTTP APIs, il n'existait pas de norme pour la réponse d'un tel point ; la plupart des applications utilisaient le seul code de statut HTTP ou leur propre format. Le projet proposait un corps JSON de type `application/health+json` avec un champ obligatoire `status` valant `pass`, `fail` ou `warn` : `pass` et `warn` accompagnent un code 2xx ou 3xx, et `fail` un 4xx ou 5xx. Sa dernière révision, d'octobre 2021, a expiré en avril 2022 ; l'IETF l'indique comme non approuvé et dépourvu de statut formel. Il est utile comme vocabulaire commun, pas comme norme.

À quoi cela ressemble en pratique

Dans un Dockerfile, le contrôle accompagne l'image qu'il vérifie. Ici, le petit programme de sonde de l'application demande un chemin sur localhost et se termine avec 0 ou 1 :

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

Docker exécute la sonde toutes les 15 secondes et accorde trois secondes à chaque exécution. Les échecs de la première minute ne comptent pas, sauf si un contrôle a déjà réussi. Sur Docker Engine 25.0 ou ultérieur, les contrôles de cette minute se font toutes les cinq secondes, intervalle de démarrage prédéfini, afin de repérer sans attendre une application qui démarre vite. Après trois échecs consécutifs, le conteneur est marqué `unhealthy`. Les 4 096 premiers octets de sortie sont conservés pour `docker inspect`, et chaque changement de statut émet un événement `health_status`. La référence décrit un état, pas une action : un autre composant doit agir en conséquence.

Dans Kubernetes, on peut donner aux trois sondes des chemins distincts pour que chacune pose sa propre question :

```
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` répond pour le seul processus ; `/readyz` vérifie aussi la base de données et tout ce dont une requête ne peut se passer. La sonde de démarrage utilise le chemin de vitalité, comme le recommande Kubernetes, et accorde 24 × 5 secondes avant d'abandonner le conteneur. Une fois qu'elle réussit, la vitalité prend le relais avec une marge longue, et la disponibilité avec une courte : environ dix secondes d'échecs retirent le Pod du Service, et une minute le redémarre. Les réglages prédéfinis permettraient trois échecs espacés de dix secondes, trop peu pour une application qui a besoin de deux minutes pour charger ses données. La règle de la documentation est qu'un conteneur dont le démarrage dépasse généralement le délai initial plus failureThreshold × periodSeconds a besoin d'une sonde de démarrage.

Un contrôle de santé peut aussi être inversé. Dans la veille de l'hôte à signal de vie, de septembre 2026, un hôte de 464 MB exécutant une forge et un serveur web se vérifie toutes les quinze minutes, avec une variation aléatoire : un petit script mesure la mémoire disponible et l'espace disque libre et ne contacte un service externe que si les deux dépassent leurs seuils. Le silence est l'alerte. Un hôte à court de mémoire, privé de réseau ou qui ne démarre plus émet le même signal qu'un hôte simplement défaillant ; ce mécanisme a donc été préféré à un agent qui rapporte un statut. Le contrôle lit la mémoire disponible plutôt que libre, car le noyau remplit la mémoire inutilisée de cache. Six noms de variables contenant « free » (« libre ») ont été corrigés pour que personne ne « répare » une alarme correcte en la faisant déclencher sans cesse. La réception d'un signal confirme que le script s'est exécuté et que ses contrôles des seuils de mémoire disponible et d'espace disque libre ont réussi ; elle ne certifie ni le bon état général de l'hôte ni que le site sert des pages. La supervision peut révéler des signes avant-coureurs, dont le moment d'apparition dépend toutefois de la charge de travail et des conditions observées.

Un contrôle au démarrage répond encore à une autre question. Dans le système de courriel avec bascule et journalisation des livraisons, réalisé dans le cadre d'une activité exercée depuis 2025, le courrier passe par trois fournisseurs avec bascule automatique. En amont, un contrôle de démarrage activé par un indicateur de fonctionnalité envoie un vrai message à travers toute la pile. En développement, un échec journalise un avertissement et l'application continue ; en préproduction et en production, il est fatal, et le processus s'arrête plutôt que de déployer une version incapable d'envoyer du courrier. La page résume le résultat : toute une catégorie d'échecs silencieux passe de « un utilisateur s'en aperçoit des jours plus tard » à « le déploiement s'arrête ».

Où les problèmes surviennent

Le contrôle prouve seulement que le processus répond. Un gestionnaire qui renvoie 200 sans rien solliciter reste vert quand la base est inaccessible, le disque plein ou chaque requête réelle en erreur. Faites tester à faible coût par le contrôle de disponibilité les composants nécessaires à une requête, et ajoutez un contrôle externe suivant le parcours utilisateur par DNS, TLS et le répartiteur : le kubelet envoie ses sondes directement à l'adresse IP du Pod et ne voit rien de cela.

La vitalité dépend d'un élément extérieur au conteneur. Si sa sonde interroge la base de données, une base lente la fait échouer sur toutes les répliques à la fois, et Kubernetes les redémarre au moment où le service est déjà en difficulté. Faites répondre la vitalité pour le seul processus, comme le conseille Kubernetes, et placez les dépendances dans la disponibilité, où un échec détourne le trafic au lieu de tuer les conteneurs.

Les délais ne correspondent pas à l'application. Sans autre réglage, Kubernetes attend une seconde : un point de terminaison qui ralentit sous charge fait échouer la sonde alors que l'application est simplement occupée. Un conteneur qui démarre plus lentement que ne l'autorise sa sonde de vitalité est tué avant d'être prêt, encore et encore. Mesurez le temps de réponse sous charge et fixez un délai supérieur ; donnez aux applications lentes au démarrage une sonde dimensionnée pour leur démarrage réel le plus lent.

Une redirection réussit le contrôle. Une sonde HTTP Kubernetes compte tout statut de 200 à 399 comme un succès : un chemin qui redirige vers une page de connexion paraît donc sain. Si la redirection vise un autre hôte, le kubelet ne la suit pas et note la sonde comme réussie, avec un avertissement. Avec le schéma HTTPS, il omet aussi la vérification du certificat ; un certificat expiré ne fait donc jamais échouer une sonde. Faites pointer les sondes vers un chemin qui répond directement 200 et renvoie un 5xx en cas de problème, et vérifiez les certificats depuis l'extérieur.

Le point de santé en dit trop à trop de monde. Une réponse détaillée qui énumère chaque dépendance, version et nom d'hôte renseigne autant un attaquant qu'un répartiteur. Les considérations de sécurité de ce document de travail précisent que ces points peuvent nécessiter une authentification et un contrôle d'accès fondé sur les rôles. Limitez la réponse publique à un statut, et ne servez les détails que sur un port interne ou à des appelants authentifiés.

Le contrôle coûte plus qu'il ne devrait. Kubernetes prévient que les sondes `exec` créent des processus à chaque exécution, ce qui peut ajouter une charge CPU sur des nœuds densément occupés avec des intervalles courts. Une sonde qui exécute une requête lourde toutes les quelques secondes sur chaque réplique constitue elle-même une charge. Préférez les sondes HTTP, TCP ou gRPC, gardez le point de terminaison peu coûteux et mettez quelques secondes en cache le résultat d'un contrôle de dépendance onéreux au lieu de le répéter pour chaque sonde.

Pour aller plus loin

- Référence Dockerfile : HEALTHCHECK — Instruction, options, valeurs prédéfinies et sens de chaque code de sortie.
- Sondes de vitalité, de disponibilité et de démarrage — Rôle de chaque sonde, quatre mécanismes et tous les champs de configuration avec leurs valeurs prédéfinies.
- Configurer les sondes de vitalité, de disponibilité et de démarrage — Exemples complets, dont une sonde de démarrage accordant cinq minutes à une application lente.
- Health Check Response Format for HTTP APIs — Document de travail Internet-Draft expiré à l'origine de `application/health+json` et de ses statuts pass, fail et warn.
- Utiliser nginx comme répartiteur HTTP — Contrôles passifs dans le trafic réel de nginx open source et réglages `max_fails` et `fail_timeout`.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le parcours de déploiement, les risques opérationnels et les personnes d'astreinte. Nous pouvons définir un changement avec des vérifications, une procédure de reprise et une responsabilité claire.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce qu'un contrôle de santé ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

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

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/health-check/