Engineer

ヘルスチェック

サービスが正常に機能するかを自動で確かめる検査

どのようなものか

ヘルスチェックは、稼働中のサービスが本来の役割を果たせるか、小さいテストを定期的に繰り返し、果たせないときに何をするかも決める仕組みです。専用パスへのHTTPリクエスト、コンテナー内のコマンド、TCPポートへの接続試行などを使います。測定を制御に変えるのは、その規則です。失敗すると、配信先からサーバーを外す、コンテナーを再起動する、デプロイを止める、誰かを呼び出す、といった処理につなげられます。

Dockerは、この仕組みをイメージへ組み込みます。Dockerfileの `HEALTHCHECK` が、コンテナー内で一定間隔で動かすコマンドを指定します。終了ステータス0は正常、1は異常で、2は予約済みなので使ってはいけません。検査付きのコンテナーは `starting` から始まり、一回でも通ると `healthy` 、指定回数連続で失敗すると `unhealthy` になります。既定は30秒ごと、タイムアウト30秒、再試行三回です。時間切れは失敗に数え、検査コマンドのプロセスを終了させます。任意の起動猶予期間は、遅いアプリに起動時間を与え、その間の失敗は数えません。Dockerfileに置く命令は一つで、複数あれば最後が勝ち、 `HEALTHCHECK NONE` は、基底イメージから継承した検査を止めます。

Kubernetesは、結果の違う三つのプローブへ分けます。起動プローブ（startup）は起動が終わったかを問い、成功まで残り二つを待たせます。失敗し続ければコンテナーを終了し、再起動ポリシーに従います。生存確認プローブ（liveness）は再起動すべきかを問い、プロセスが動いても進まない、デッドロックなどに使います。準備状態プローブ（readiness）は、トラフィックを受けるべきかを問います。失敗しても再起動せず、一致するすべてのServiceのエンドポイントからPodのアドレスを外し、再び通るまで、ほかのレプリカへ送ります。

各プローブは四つの仕組みの一つを使います。 `httpGet` は、200以上400未満の状態で合格します。 `tcpSocket` は、すぐ閉じても、ポートが接続を受ければ通ります。 `grpc` は、サービスのgRPCヘルスチェックが `SERVING` と答えれば通りま

す。 `exec` は、コンテナー内でコマンドを動かし、終了ステータス0で通ります。指定しなければ10秒ごとに調べ、一秒待ち、三回続けて失敗すると対処します。

ロードバランサーには、もう一種類あります。これまでのものは能動型で、利用者がいてもいなくても、テストリクエストを送ります。受動型は、実際の通信を見ます。オープンソースのnginxは、上流サーバーへのリクエストがエラーに終わると、そのサーバーを異常とみなし、しばらく避けます。 `max_fails` の既定値1なら、一度の失敗で十分です。nginxの文書は、アプリケーションのヘルスチェックを、有料のNGINX Plusの機能に挙げます。受動型に追加の負担はありませんが、故障を知るのは利用者のリクエストが失敗した後で、通信のないサーバーについては何も分かりません。

概念を判断に生かす

「ヘルスチェックとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 障害が小さいうちに知らせる監視とアラートを整え、安定した状態をどれだけ維持できているかを数値で示します。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 社内初のオブザーバビリティダッシュボードを構築 — 社内システムの状態を初めて可視化

社内初のオブザーバビリティダッシュボードを構築 お問い合わせ →

なぜ重要なのか

プロセスが稼働していることと、サービスが正常に機能することは別です。Dockerの参照文書は、典型例を示します。ウェブサーバーが無限ループへ入り、新しい接続を受けなくても、プロセスは残ります。再起動ポリシーのあるsystemdユニットなど、存在だけを見るものには、異常は見えません。利用者と同じ入口から尋ねて初めて区別できます。ヘルスチェックの意義は、「動いているか」を「応答するか」に変えることです。

二つの決定で、有用にも危険にもなります。一つ目は検査内容で、「正常に機能する」の定義をコードで書くことです。何も触らず200を返すハンドラーは、ウェブのフレームワークが動くとだけ証明します。データベースにも問い合わせれば、到達可能だとも証明し、届かなければ失敗します。二つ目は失敗後の処理で、影響範囲を決めます。一レプリカを配信先から外すのは軽く、戻せます。再起動すれば、処理中の仕事もすでにデータを蓄えたキャッシュも失います。すべてが同時に同じ検査で落ち、全部再起動すれば、遅い依存先を全面停止へ変えます。

Kubernetesも、まさにこの問題を警告します。文書は、livenessは、本当に復旧不能な障害を示すよう設定する必要があり、誤実装は、高負荷中の再起動、クライアントの失敗、残ったPodへの負荷集中という、連鎖障害を起こし得るとします。続く助言は、二つの問いを分けることです。バックエンドが不可欠なアプリでも、自身が正常な間はlivenessを通し、readinessでは必要な各バックエンドも調べます。エラーしか返せないPodをトラフィックが避け、別の場所の故障でPodを終了させずに済みます。

監視にも使えます。ロードバランサーが定期的に尋ねるエンドポイントは、外からも調べられ、その失敗のアラートが、インシデントの最初の兆候になることもよくあります。Irakli NadareishviliがInternet-DraftのHTTP API向けヘルスチェック応答形式を書いた時点では、戻すものの標準はなく、大半のアプリが状態コードだけか独自形式に頼っていました。草案は、メディア型 `application/health+json` のJSON本文に、必須の `status` として `pass` 、 `fail` 、 `warn` を持たせる案です。 `pass` と `warn` は2xxか3xx、 `fail` は4xxか5xxを伴います。最終改訂は2021年10月、失効は2022年4月で、IETFは承認しておらず、正式な地位はないと記します。共通語彙には有用でも、標準ではありません。

実際の活用例

Dockerfileなら、検査対象のイメージ自体に検査を同梱できます。ここでは、アプリ自身の小さいプローブプログラムがlocalhostのパスへリクエストを送り、0か1で終わります。

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

15秒ごとに動かし、一回に三秒を与えます。最初の一分の失敗は、それまでに通った検査がなければ、数えません。Docker Engine 25.0以降では、この一分間は、既定の起動時の間隔である五秒ごとに調べ、速い起動にもすぐ気付きます。連続三回の失敗で `unhealthy` にします。出力の最初の4096バイトを `docker inspect` 用に残し、状態が変わるたび `health_status` イベントを出します。文書が記すのは状態で、動作ではありません。対処は別のものが行う必要があります。

Kubernetesでは三つに別々のパスを与え、それぞれの問いを尋ねられます。

```
containers:
  - name: api
    ports:
```

```
      - containerPort: 8080
    startupProbe:
      httpGet: { path: /livez, port: 8080 }
      periodSeconds: 5
      failureThreshold: 24 # 起動を最大2分待つ
    livenessProbe:
      httpGet: { path: /livez, port: 8080 }
      periodSeconds: 10
      timeoutSeconds: 2
      failureThreshold: 6 # 再起動まで約1分
    readinessProbe:
      httpGet: { path: /readyz, port: 8080 }
      periodSeconds: 5
      timeoutSeconds: 2
      failureThreshold: 2
```

`/livez` はプロセスだけ、 `/readyz` はデータベースや、要求に不可欠なものも調べます。startupは、文書が勧めるとおりlivenessのパスを使い、24 × 5秒まで待ってから諦めます。通れば、livenessは長く、readinessは短く待って対処します。約十秒の失敗でServiceから外し、一分で再起動します。既定では十秒間隔の三回なので、データ読み込みに二分必要なアプリには短すぎます。文書の規則は、起動に通常、初期遅延とfailureThreshold × periodSecondsの合計を超える時間がかかるコンテナーには、startupが必要というものです。

反対向きの検査もできます。2026年9月のデッドマンズスイッチによるホスト監視では、フォージとウェブサーバーを動かす464 MBのホストが、時間に揺らぎを加えて十五分ごとに自身を調べます。小さいスクリプトが、利用可能なメモリーとディスク空き容量を測り、両方とも閾値を超えるときだけ外部サービスへ通知します。沈黙がアラートです。メモリー不足、ネットワーク喪失、起動不能も、単に正常性に問題がある場合と同じ信号になるため、状態を報告するエージェントより、この形を選びました。カーネルが空いたメモリーをキャッシュで埋めるので、freeでなくavailableを読みます。「free」とした六つの変数名も直し、正常な警報を「修正」して、常に発報するものにしないようにしました。信号の受信は、スクリプトが動作し、利用可能メモリーと空きディスク容量が設定した閾値の条件を満たしたことを確認します。ホスト全体の健全性や、サイトが配信できていることを証明するものではありません。監視によって兆候を捉えられますが、その時期は負荷や観測する条件によって異なります。

起動時の検査は、また別の問いです。2025年からの担当期間に含まれるフェイルオーバーと配信ログを備えるメールの事例では、三事業者を自動切り替えで使い、その前に、フラグで有効にする起動検査が、全構成を通して実際のメッセージを送ります。開発では失敗を警告として記録して続けますが、ステージングと

本番では致命的で、送れないビルドをデプロイせず終了します。事例は、気付かれない失敗の一種類全体を、「数日後に利用者が気付く」段階から「デプロイが止まる」段階へ移したとまとめます。

問題が起きるところ

プロセスの応答しか証明しません。 何も触らず200を返すハンドラーは、データベースに届かない場合も、ディスクが満杯の場合も、すべての実際のリクエストが失敗する場合も、正常を示します。readinessで、リクエストに必要なものを低い負荷で試し、DNS、TLS、ロードバランサーを通る利用者の経路を、外からも調べます。kubeletはPodのIPへ直接送り、それらは見えないためです。

livenessが、コンテナー外へ依存します。 データベースを尋ねると、遅いだけで全レプリカが同時に失敗し、すでに苦しい瞬間に、全部再起動します。文書に従い、livenessは自身だけを問い、依存先はreadinessへ入れます。失敗時に終了させず、通信先を変えられます。

時間がアプリに合いません。 既定は一秒待つだけなので、高負荷で応答が遅れると、アプリが忙しいだけでもプローブは失敗します。起動がlivenessの許す時間より遅ければ、準備前に何度も終了させられます。負荷中に測って、それより長いタイムアウトを与え、遅い起動には、実際の最遅時間に合わせたstartupを付けます。

リダイレクトが通ります。 KubernetesのHTTPプローブは200から399を成功とし、ログインページへ転送するパスも正常と判定されます。kubeletは別ホストへの転送を追わず、警告付きで成功と記録します。HTTPSでも証明書を検証しないので、期限切れでは失敗しません。直接200を返し、問題時は5xxのパスを使い、証明書は外から調べます。

正常確認の入口が、多くの人へ詳しく語りすぎます。 全依存先、版、ホスト名を列挙する詳しい応答は、ロードバランサーだけでなく攻撃者にも情報を与えます。Internet-Draftのセキュリティ節も、認証とロールによるアクセス制御が必要かもしれないと記します。公開するのは状態だけにし、詳細は内部ポートか、認証済みの呼び出し元だけへ返します。

検査自体が重すぎます。 Kubernetesは、`exec` が毎回プロセスをforkし、コンテナーを多く詰めたノードで、短い間隔で動かすと、CPU負担が増え得ると警告します。全レプリカで数秒ごとに重いクエリを動かす検査も、それ自体が負荷で

す。HTTP、TCP、gRPCを優先し、入口を軽くし、負担の大きい依存先検査の結果は数秒キャッシュして、毎回繰り返さないようにします。

参考資料

- Dockerfile参照文書：HEALTHCHECK — 命令、選択肢と既定値、各終了ステータスの意味です。
- 生存確認・準備状態・起動の各プローブ — 三種類の働き、四つの検査方法、全設定項目と既定値です。
- 生存確認・準備状態・起動の各プローブの設定 — 遅いアプリに五分を与えるstartupなど、具体例です。
- HTTP API向けヘルスチェック応答形式 — `application/health+json` とpass、fail、warnの出典である、失効したInternet-Draftです。
- nginxをHTTPロードバランサーとして使う — オープンソース版の、実通信による受動的検査と、 `max_fails` 、 `fail_timeout` の設定です。

用語集

実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「ヘルスチェックとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く 連絡先カード

心を込めて丁寧に制作： D.

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

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/health-check/