Engineer

Hälsokontroll

ett automatiskt test av om en tjänst fungerar

Vad det är

En hälsokontroll är ett litet test som upprepas enligt ett schema och frågar en körande tjänst om den kan göra sitt jobb, tillsammans med en regel för vad som händer när svaret är nej. Testet kan vara en HTTP-förfrågan till en särskild sökväg, ett kommando som körs inuti en container eller ett försök att öppna en TCP-port. Det är regeln som gör kontrollen till mer än en mätning: en misslyckad hälsokontroll kan ta en server ur rotation, starta om en container, stoppa en driftsättning eller väcka någon mitt i natten.

Docker bygger in idén i själva imagen. Instruktionen `HEALTHCHECK` i en Dockerfile anger ett kommando som Docker kör inuti containern med ett fast intervall. Slutkod 0 betyder att containern är frisk, 1 att den inte är det, och 2 är reserverad och får inte användas. En container med en kontroll börjar i tillståndet `starting`, blir `healthy` så snart en kontroll går igenom och övergår till `unhealthy` efter ett bestämt antal misslyckanden i rad. Standardvärdena är en kontroll var 30:e sekund, en tidsgräns på 30 sekunder och tre försök; en körning som överskrider sin tidsgräns räknas som ett misslyckande, och dess process avslutas med tvång. En valfri startperiod ger ett långsamt program tid att starta, och misslyckanden under den räknas inte. En Dockerfile har en sådan instruktion, den sista gäller om det finns flera, och `HEALTHCHECK NONE` stänger av en kontroll som ärvts från basimagen.

Kubernetes delar upp frågan i tre prober, var och en med sin egen konsekvens. En startup-prob frågar om programmet har startat klart; tills den lyckas väntar de två andra, och om den fortsätter att misslyckas avslutas containern och hanteras enligt sin omstartspolicy. En liveness-prob frågar om containern ska startas om och är avsedd för tillstånd som ett dödläge (deadlock), där processen kör men inte kommer framåt. En readiness-prob frågar om containern ska få trafik. När

den misslyckas startas ingenting om; i stället tas Podens adress bort från endpoints för varje Service som väljer den, och förfrågningarna går till de andra replikerna tills proben går igenom igen.

Varje prob använder en av fyra mekanismer. En `httpGet` -prob går igenom på varje statuskod från och med 200 till och med 399. En `tcpSocket` -prob går igenom om porten tar emot en anslutning, även en som stängs direkt. En `grpc` -prob går igenom när tjänstens hälsokontroll i gRPC svarar `SERVING` . En `exec` -prob kör ett kommando i containern och går igenom på slutkod 0. Om inget annat anges kontrollerar Kubernetes var tionde sekund, väntar en sekund på svar och agerar efter tre misslyckanden i rad.

Lastbalanserare lägger till en andra sort. Kontrollerna ovan är aktiva: något skickar en testförfrågan oavsett om det finns användare eller inte. En passiv kontroll följer i stället den verkliga trafiken. Så fungerar nginx med öppen källkod: när en förfrågan till en upstream-server slutar med ett fel markerar nginx servern som felande och undviker den en stund, och med standardvärdet 1 för `max_fails` räcker ett enda misslyckat försök. Enligt dokumentationen för nginx hör hälsokontroller av applikationer (application health checks) till funktionerna i de betalda NGINX Plus-prenumerationerna. En passiv kontroll kostar inget extra, men den får veta att en server är trasig genom en användares misslyckade förfrågan, och den säger ingenting om en server som inte får någon trafik.

Använd begreppet i ett beslut

Använd ”Vad är en hälsokontroll?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Övervakning och larm som säger till medan ett fel fortfarande är litet, och siffrorna som visar hur ofta det förblir så.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Byggde företagets första observerbarhetsdashboard — företagets första överblick över de egna systemen

Byggde företagets första observerbarhetsdashboard Hör av er →

Varför det spelar roll

En process som kör är inte detsamma som en tjänst som fungerar. Dockers referens nämner det typiska fallet: en webbserver som har fastnat i en oändlig loop och slutat ta emot anslutningar medan processen fortfarande lever. Allt som bara håller koll på om processen finns, till exempel en systemd-enhet med

omstartspolicy, ser inget fel. Bara en fråga som ställs genom samma dörr som användarna går in genom kan se skillnaden, och det är hela poängen med en hälsokontroll: den byter ut ”kör den?” mot ”svarar den?”.

Två beslut avgör om en kontroll är nyttig eller farlig. Det första är vad kontrollen testar, vilket i praktiken är en definition av ”fungerar”, skriven i kod. En hanterare som returnerar 200 utan att röra något bevisar att webbramverket är uppe; en hanterare som också frågar databasen bevisar att databasen går att nå, och misslyckas när den inte gör det. Det andra är vad som händer vid ett misslyckande, och det avgör hur långt skadan når. Att ta en replik ur rotation är billigt och går att ångra. Att starta om den kastar bort det arbete den höll på med och dess varma cachar, och om alla repliker misslyckas med samma kontroll samtidigt gör en omstart av alla ett långsamt beroende till ett totalt driftstopp.

Kubernetes varnar för just detta. Dokumentationen säger att liveness-prober måste ställas in så att de verkligen visar ett fel som programmet inte kan återhämta sig från på egen hand, och att en felaktig implementering kan orsaka kaskadfel: containrar som startas om under hög belastning, klientförfrågningar som misslyckas och mer arbete för de poddar som är kvar. Rådet som följer är att hålla isär de två frågorna. När ett program är strikt beroende av bakomliggande tjänster ska dess liveness-prob gå igenom så länge programmet självt är friskt, medan readiness-proben också kontrollerar varje nödvändig tjänst bakom. Då undviker trafiken en Pod som bara kunde ha svarat med fel, och Poden dödas inte för ett fel som ligger någon annanstans.

Hälsokontroller kommer också övervakningen till godo. En endpoint som en lastbalanserare frågar kan också frågas utifrån, och det larm som går när den fallerar är ofta det första tecknet på en incident. När Irakli Nadareishvili skrev Health Check Response Format for HTTP APIs, ett Internet-Draft inom IETF, fanns det ingen standard för vad en sådan endpoint ska returnera, och de flesta program nöjde sig med HTTP-statuskoden eller använde ett eget format. Utkastet föreslog ett JSON-svar med medietypen `application/health+json` och ett obligatoriskt `status`-fält med värdet `pass`, `fail` eller `warn`, där `pass` och `warn` hör ihop med en statuskod i 2xx eller 3xx och `fail` med en i 4xx eller 5xx. Den sista versionen, från oktober 2021, löpte ut i april 2022, och IETF anger att den varken är godkänd av IETF eller har någon formell status. Den är användbar som ett gemensamt ordförråd, inte som en standard.

Så ser det ut i praktiken

I en Dockerfile följer kontrollen med den image den kontrollerar. Här frågar programmets eget lilla probprogram efter en sökväg på localhost och avslutar med 0 eller 1:

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

Docker kör proben var 15:e sekund och ger varje körning tre sekunder. Misslyckanden under den första minuten räknas inte, om inte en kontroll redan har gått igenom, och i Docker Engine 25.0 eller senare körs kontrollerna under den minuten var femte sekund, vilket är standardintervallet under startperioden, så att en snabb start märks snabbt. Efter tre misslyckanden i rad markeras containern som `unhealthy`. De första 4096 byten av probens utdata sparas för `docker inspect`, och varje statusändring skickar ut en `health_status`-händelse. Referensen beskriver ett tillstånd, inte en åtgärd: något annat måste agera på det.

I Kubernetes kan de tre proberna få var sin sökväg, så att var och en ställer sin egen fråga:

```
containers:
  - name: api
    ports:
      - containerPort: 8080
    startupProbe:
      httpGet: { path: /livez, port: 8080 }
      periodSeconds: 5
      failureThreshold: 24 # upp till 2 minuter att starta
    livenessProbe:
      httpGet: { path: /livez, port: 8080 }
      periodSeconds: 10
      timeoutSeconds: 2
      failureThreshold: 6 # ungefär en minut före omstart
    readinessProbe:
      httpGet: { path: /readyz, port: 8080 }
      periodSeconds: 5
      timeoutSeconds: 2
      failureThreshold: 2
```

`/livez` svarar bara för själva processen, medan `/readyz` också kontrollerar databasen och allt annat som en förfrågan inte klarar sig utan. Startup-proben använder liveness-sökvägen, som dokumentationen för Kubernetes rekommenderar, och ger 24 × 5 sekunder innan containern ges upp. När den har gått igenom tar liveness över med lång stubin och readiness med kort: ungefär tio sekunder av misslyckanden tar Poden ur Servicen, och en minut av dem startar om den. Standardvärdena skulle ge tre misslyckanden med tio sekunders mellanrum, och det är för lite för ett program som behöver två minuter för att läsa in sina data. Dokumentationens egen tumregel är att en container som vanligtvis tar längre tid att starta än initialDelaySeconds plus failureThreshold × periodSeconds behöver en startup-prob.

En hälsokontroll kan också vändas ut och in. I värdövervakning med dödmansgrepp, från september 2026, kontrollerar en värd på 464 MB som kör en git-forge och en webbserver sig själv var femtonde minut, med slumpmässig förskjutning: ett litet skript mäter tillgängligt minne och ledigt diskutrymme och pingar en extern tjänst bara om båda ligger över sina tröskelvärden. Tystnaden är larmet. En värd som har fått slut på minne, tappat nätverket eller slutat starta ger samma signal som en som bara inte är frisk, och därför valdes den här formen framför en agent som rapporterar en status. Kontrollen läser tillgängligt minne och inte ledigt minne, eftersom kärnan använder outnyttjat minne som cache, och sex variabelnamn som sa ”free” rättades, så att ingen skulle ”laga” ett fungerande larm till ett som löser ut permanent. En mottagen signal bekräftar att skriptet kördes och att dess tröskelkontroller för tillgängligt minne och ledigt diskutrymme godkändes; den intygar inte värdens allmänna tillstånd eller att webbplatsen svarar. Övervakning kan visa varningssignaler, men tidpunkten beror på belastningen och de förhållanden som observeras.

En kontroll vid uppstart besvarar ännu en fråga. I e-post med failover och leveransloggning, från 2025, går posten genom tre leverantörer med automatisk failover, och framför alltihop skickar en kontroll vid uppstart ett verkligt meddelande genom hela stacken, bakom en flagga. I utveckling loggar ett misslyckande en varning och programmet fortsätter; i staging och produktion är felet fatalt, och processen avslutas i stället för att ett bygge som inte kan skicka e-post driftsätts. Sidan sammanfattar resultatet så här: en hel kategori av tysta fel flyttades från ”en användare märker det dagar senare” till ”driftsättningen stoppas”.

Var det går fel

Kontrollen bevisar bara att processen svarar. En hanterare som returnerar 200 utan att röra något förblir grön medan databasen inte går att nå, disken är full eller varje verklig förfrågan slutar med ett fel. Låt readiness-kontrollen pröva det en förfrågan behöver, på ett billigt sätt, och lägg till en kontroll utifrån som följer användarens väg genom DNS, TLS och lastbalanseraren, eftersom kubelet skickar sina prober direkt till Podens egen IP-adress och inte ser något av det.

Liveness beror på något utanför containern. Om liveness-proben frågar databasen får en långsam databas den att misslyckas i alla repliker på en gång, och Kubernetes startar om dem alla precis när tjänsten redan har det svårt. Låt liveness bara svara för själva processen, som dokumentationen för Kubernetes råder, och lägg beroendena i readiness, där ett misslyckande leder bort trafiken i stället för att döda containrar.

Tidsinställningarna passar inte programmet. Kubernetes väntar som standard en sekund på svar, så en endpoint som blir långsammare under belastning misslyckas medan programmet bara har mycket att göra, och en container som startar långsammare än dess liveness-prob tillåter dödas innan den är klar, om och om igen. Mät hur lång tid endpointen tar under belastning och sätt tidsgränsen över det, och ge långsamma program en startup-prob som är dimensionerad efter den långsammaste verkliga starten.

En omdirigering går igenom. En HTTP-prob i Kubernetes räknar varje status från 200 till 399 som en framgång, så en sökväg som omdirigerar till en inloggningssida ser frisk ut. Pekar omdirigeringen till en annan värd följer kubelet den inte, utan registrerar proben som lyckad med en varning. Med HTTPS hoppar kubelet dessutom över certifikatkontrollen, så ett utgånget certifikat får aldrig en prob att misslyckas. Rikta proberna mot en sökväg som svarar 200 direkt och returnerar en 5xx när något är fel, och kontrollera certifikaten utifrån.

Endpointen säger för mycket till för många. Ett detaljerat svar som räknar upp varje beroende, version och värdnamn informerar en angripare lika väl som en lastbalanserare, och säkerhetsavsnittet i utkastet påpekar att endpoints för hälsokontroller kan behöva autentisering och rollbaserad åtkomstkontroll. Begränsa det offentliga svaret till en status, och visa detaljerna bara på en intern port eller för autentiserade klienter.

Kontrollen kostar mer än den borde. Kubernetes varnar för att `exec` -prober startar nya processer vid varje körning, vilket på tätt packade noder med korta intervall kan öka CPU-belastningen på noden, och en prob som kör en tung fråga med några sekunders mellanrum på varje replik är en belastning i sig. Föredra HTTP-, TCP- eller gRPC-prober, håll endpointen billig och cacha resultatet av en dyr beroendekontroll i några sekunder i stället för att upprepa den för varje prob.

Läs mer

- Dockerfile reference: HEALTHCHECK — instruktionen, dess alternativ och standardvärden och vad varje slutkod betyder.
- Liveness, Readiness, and Startup Probes — vad varje probtyp gör, de fyra kontrollmekanismerna och alla konfigurationsfält med sina standardvärden.
- Configure Liveness, Readiness and Startup Probes — genomgångna exempel, bland annat en startup-prob som ger ett långsamt program fem minuter.
- Health Check Response Format for HTTP APIs — det utgångna Internet-Draftet bakom `application/health+json` och dess statusvärden pass, fail och warn.
- Using nginx as HTTP load balancer — de passiva hälsokontrollerna i nginx med öppen källkod, som sker i själva trafiken (in-band), och inställningarna `max_fails` och `fail_timeout` .

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är en hälsokontroll?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

welcome@engineer.company +45 50 10 61 84 Köpenhamn, Danmark

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/health-check/