Engineer

Helsesjekk

en automatisk test av om en tjeneste fungerer

Hva det er

En helsesjekk er en liten test som gjentas etter en fast plan og spør en kjørende tjeneste om den kan gjøre jobben sin, sammen med en regel for hva som skjer når svaret er nei. Testen kan være en HTTP-forespørsel til en egen sti, en kommando som kjøres inne i en container, eller et forsøk på å åpne en TCP-port. Det er regelen som gjør sjekken til mer enn en måling: en mislykket helsesjekk kan ta en server ut av rotasjon, starte en container på nytt, stanse en produksjonssetting eller vekke noen midt på natten.

Docker bygger ideen inn i selve imaget. Instruksjonen `HEALTHCHECK` i en Dockerfile angir en kommando som Docker kjører inne i containeren med et fast intervall. Avslutningskode 0 betyr frisk, 1 betyr usunn, og 2 er reservert og skal ikke brukes. En container med en sjekk starter i tilstanden `starting`, blir `healthy` så snart én sjekk går gjennom, og går over til `unhealthy` etter et fastsatt antall feil på rad. Standardverdiene er en sjekk hvert 30. sekund, et tidsavbrudd på 30 sekunder og tre forsøk; en kjøring som overskrider tidsavbruddet, teller som en feil, og prosessen blir drept. En valgfri startperiode gir et tregt program tid til å starte, og feil innenfor den teller ikke. En Dockerfile har én slik instruksjon, den siste gjelder hvis det står flere, og `HEALTHCHECK NONE` slår av en sjekk som er arvet fra basisimaget.

Kubernetes deler spørsmålet i tre prober, hver med sin egen konsekvens. En startup-probe spør om programmet er ferdig med å starte; til den lykkes, venter de to andre, og fortsetter den å feile, blir containeren drept og håndtert etter omstartspolicyen sin. En liveness-probe spør om containeren bør startes på nytt, og er ment for tilstander som en vranglås (deadlock), der prosessen kjører, men ikke kommer videre. En readiness-probe spør om containeren skal få trafikk. Når den feiler, blir ingenting startet på nytt; i stedet fjernes adressen til Pod-en fra endepunktene til hver Service som velger den, og forespørslene går til de andre replikaene til proben går gjennom igjen.

Hver probe bruker én av fire mekanismer. En `httpGet` -probe går gjennom på enhver statuskode fra og med 200 til og med 399. En `tcpSocket` -probe går gjennom hvis porten tar imot en tilkobling, også en som lukkes med en gang. En `grpc` -probe går gjennom når tjenestens helsesjekk i gRPC svarer `SERVING` . En `exec` -probe kjører en kommando i containeren og går gjennom på avslutningskode 0. Om ikke annet er angitt, prober Kubernetes hvert tiende sekund, venter ett sekund på svar og handler etter tre feil på rad.

Lastbalanserere legger til en annen sort. Sjekkene ovenfor er aktive: noe sender en testforespørsel enten det er brukere der eller ikke. En passiv sjekk følger i stedet med på den virkelige trafikken. Slik fungerer nginx med åpen kildekode: når en forespørsel til en upstream-server ender med en feil, merker nginx serveren som feilet og unngår den en stund, og med standardverdien 1 for `max_fails` er ett mislykket forsøk nok. Ifølge dokumentasjonen til nginx hører helsesjekker av applikasjoner (application health checks) til funksjonene i de betalte NGINX Plus-abonnementene. En passiv sjekk koster ingenting ekstra, men den får vite om en ødelagt server gjennom en brukers mislykkede forespørsel, og den sier ingenting om en server som ikke får trafikk.

Bruk begrepet i en beslutning

Bruk «Hva er en helsesjekk?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Overvåking og varsler som sier fra mens en feil fortsatt er liten, og tallene som viser hvor ofte den holder seg liten.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Bygde selskapets første observerbarhetsdashbord — selskapets første innsyn i egne systemer

Bygde selskapets første observerbarhetsdashbord Ta kontakt →

Hvorfor det er viktig

En prosess som kjører, er ikke det samme som en tjeneste som virker. Referansen til Docker nevner det typiske eksempelet: en webserver som har hengt seg i en uendelig løkke og sluttet å ta imot tilkoblinger, mens prosessen fortsatt lever. Alt som bare følger med på om prosessen finnes, som en systemd-enhet med omstartspolicy, ser ingenting galt. Bare et spørsmål som stilles gjennom den samme døren som brukerne går inn gjennom, kan se forskjellen, og det er hele begrunnelsen for en helsesjekk: den bytter ut «kjører den?» med «svarer den?».

To beslutninger avgjør om en sjekk er nyttig eller farlig. Den første er hva sjekken tester, og det er i praksis en definisjon av «virker», skrevet i kode. En handler som returnerer 200 uten å røre noe, beviser at webrammeverket er oppe; en handler som også spør databasen, beviser at databasen kan nås, og feiler når den ikke kan det. Den andre er hva som skjer ved feil, og den avgjør hvor langt skaden rekker. Å ta én replika ut av rotasjon er billig og kan gjøres om. Å starte den på nytt kaster bort arbeidet den holdt på med og de varme cachene, og hvis alle replikaene feiler den samme sjekken samtidig, gjør en omstart av alle sammen en treg avhengighet om til et fullstendig driftsavbrudd.

Kubernetes advarer nettopp mot dette. Dokumentasjonen sier at liveness-prober må settes opp slik at de virkelig viser en feil programmet ikke kommer seg av på egen hånd, og at en feil implementering kan gi kaskadefeil: containere som startes på nytt under høy last, klientforespørsler som feiler, og mer arbeid på podene som er igjen. Rådet som følger, er å holde de to spørsmålene fra hverandre. Når et program er strengt avhengig av bakenforliggende tjenester, skal liveness-proben gå gjennom så lenge programmet selv er friskt, mens readiness-proben også sjekker hver nødvendig tjeneste bak. Da unngår trafikken en Pod som bare kunne svart med feil, og Pod-en blir ikke drept for en feil som ligger et annet sted.

Helsesjekker kommer også overvåkingen til gode. Et endepunkt som en lastbalanserer spør, kan også spørres utenfra, og driftsvarselet som går når det feiler, er ofte det første tegnet på en driftshendelse. Da Irakli Nadareishvili skrev Health Check Response Format for HTTP APIs, et Internet-Draft i IETF, fantes det ingen standard for hva et slikt endepunkt skal returnere, og de fleste programmer nøyde seg med HTTP-statuskoden eller brukte et eget format. Utkastet foreslo et JSON-svar med medietypen `application/health+json` og et obligatorisk `status`-felt med verdien `pass`, `fail` eller `warn`, der `pass` og `warn` hører sammen med en statuskode i 2xx eller 3xx, og `fail` med en i 4xx eller 5xx. Den siste versjonen, fra oktober 2021, utløp i april 2022, og IETF oppgir at den verken er godkjent av IETF eller har formell status. Den er nyttig som et felles vokabular, ikke som en standard.

Slik ser det ut i praksis

I en Dockerfile følger sjekken med imaget den sjekker. Her spør programmets eget lille probeprogram etter en sti på localhost og avslutter 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 kjører proben hvert 15. sekund og gir hver kjøring tre sekunder. Feil i løpet av det første minuttet teller ikke med mindre en sjekk allerede har gått gjennom, og på Docker Engine 25.0 eller nyere kjøres sjekkene i det minuttet hvert femte sekund, som er standardintervallet i startperioden, slik at en rask oppstart blir oppdaget raskt. Etter tre feil på rad merkes containeren som `unhealthy`. De første 4096 bytene av utdataene fra proben lagres for `docker inspect`, og hver statusendring sender ut en `health_status`-hendelse. Referansen beskriver en status, ikke en handling: noe annet må reagere på den.

I Kubernetes kan de tre probene få hver sin sti, slik at hver av dem stiller sitt eget spørsmål:

```
containers:
  - name: api
    ports:
      - containerPort: 8080
    startupProbe:
      httpGet: { path: /livez, port: 8080 }
      periodSeconds: 5
      failureThreshold: 24 # opptil 2 minutter på å starte
    livenessProbe:
      httpGet: { path: /livez, port: 8080 }
      periodSeconds: 10
      timeoutSeconds: 2
      failureThreshold: 6 # omtrent ett minutt før omstart
    readinessProbe:
      httpGet: { path: /readyz, port: 8080 }
      periodSeconds: 5
      timeoutSeconds: 2
      failureThreshold: 2
```

`/livez` svarer bare for selve prosessen, mens `/readyz` også sjekker databasen og alt annet en forespørsel ikke klarer seg uten. Startup-proben bruker liveness-stien, slik dokumentasjonen til Kubernetes anbefaler, og gir 24 × 5 sekunder før containeren blir oppgitt. Når den har gått gjennom, tar liveness over med lang lunte og readiness med kort: rundt ti sekunder med feil tar Pod-en ut av Servicen, og ett minutt med feil starter den på nytt. Standardverdiene ville gitt tre feil med ti sekunders mellomrom, og det er for lite for et program som trenger to minutter på å laste inn dataene sine. Dokumentasjonens egen

tommelfingerregel er at en container som vanligvis bruker lengre tid på å starte enn initialDelaySeconds pluss failureThreshold × periodSeconds, trenger en startup-probe.

En helsesjekk kan også snus på hodet. I vertsovervåking med dødmannsknapp, fra september 2026, sjekker en vert på 464 MB som kjører en forge og en webserver, seg selv hvert femtende minutt, med jitter: et lite skript måler tilgjengelig minne og ledig disk og pinger en ekstern tjeneste bare når begge ligger over tersklene sine. Tausheten er varselet. En vert som har gått tom for minne, mistet nettverket eller sluttet å starte, gir det samme signalet som en som bare er usunn, og derfor ble denne formen valgt fremfor en agent som rapporterer en status. Sjekken leser tilgjengelig minne og ikke ledig minne, fordi kjernen bruker minne som ellers står ubrukt, til cache, og seks variabelnavn som sa «free», ble rettet, så ingen skulle «fikse» en alarm som virker, til en som slår ut hele tiden. Et mottatt signal bekrefter at skriptet kjørte, og at terskelkontrollene for tilgjengelig minne og ledig diskplass ble bestått; det dokumenterer ikke vertens generelle tilstand eller at nettstedet svarer. Overvåking kan avdekke varselsignaler, men tidspunktet avhenger av belastningen og forholdene som observeres.

En sjekk ved oppstart svarer på enda et spørsmål. I e-post med failover og leveringslogging, fra 2025, går e-posten gjennom tre leverandører med automatisk failover, og foran alt dette sender en oppstartskontroll en ekte melding gjennom hele stakken, styrt av et flagg. I utvikling logger en feil en advarsel, og programmet fortsetter; i staging og produksjon er feilen fatal, og prosessen avslutter heller enn å produksjonssette et bygg som ikke kan sende e-post. Siden oppsummerer resultatet slik: en hel kategori stille feil gikk fra «en bruker merker det dager senere» til «produksjonssettingen stopper».

Hvor det går galt

Sjekken beviser bare at prosessen svarer. En handler som returnerer 200 uten å røre noe, forblir grønn mens databasen er utilgjengelig, disken er full eller hver eneste ekte forespørsel ender med en feil. La readiness-sjekken prøve ut det en forespørsel trenger, på en billig måte, og legg til en sjekk utenfra som følger brukerens vei gjennom DNS, TLS og lastbalansereren, for kubelet sender probene sine rett til IP-adressen til Pod-en og ser ingenting av dette.

Liveness avhenger av noe utenfor containeren. Hvis liveness-proben spør databasen, får en treg database den til å feile i alle replikaene samtidig, og Kubernetes starter dem alle på nytt akkurat når tjenesten allerede sliter. La liveness bare svare for selve prosessen, slik dokumentasjonen til Kubernetes råder til, og legg avhengighetene i readiness, der en feil leder trafikken bort i stedet for å drepe containere.

Tidsinnstillingene passer ikke til programmet. Kubernetes venter som standard ett sekund på svar, så et endepunkt som blir tregere under last, feiler mens programmet bare har mye å gjøre, og en container som starter tregere enn liveness-proben tillater, blir drept før den er klar, igjen og igjen. Mål hvor lang tid endepunktet bruker under last, og sett tidsavbruddet over det, og gi trege programmer en startup-probe som er dimensjonert etter den tregeste reelle oppstarten.

En omdirigering går gjennom. En HTTP-probe i Kubernetes regner enhver status fra 200 til 399 som suksess, så en sti som omdirigerer til en innloggingsside, ser frisk ut. Peker omdirigeringen til en annen vert, følger ikke kubelet den, men registrerer proben som vellykket med en advarsel. Med HTTPS hopper kubelet dessuten over sertifikatkontrollen, så et utløpt sertifikat får aldri en probe til å feile. Pek probene mot en sti som svarer 200 direkte og returnerer en 5xx når noe er galt, og sjekk sertifikatene utenfra.

Endepunktet forteller for mye til for mange. Et detaljert svar som lister opp hver avhengighet, versjon og vertsnavn, informerer en angriper like godt som en lastbalanserer, og sikkerhetsavsnittet i utkastet påpeker at endepunkter for helsesjekk kan trenge autentisering og rollebasert tilgangskontroll. Begrens det offentlige svaret til en status, og vis detaljene bare på en intern port eller til autentiserte klienter.

Sjekken koster mer enn den burde. Kubernetes advarer om at `exec` -prober starter nye prosesser ved hver kjøring, noe som på tettpakkede noder med korte intervaller kan øke CPU-belastningen på noden, og en probe som kjører en tung spørring med noen sekunders mellomrom på hver replika, er en belastning i seg selv. Foretrekk HTTP-, TCP- eller gRPC-prober, hold endepunktet billig, og mellomlagre resultatet av en dyr avhengighetssjekk i noen sekunder i stedet for å gjenta den for hver probe.

Les mer

- Dockerfile reference: HEALTHCHECK — instruksjonen, valgene og standardverdiene, og hva hver avslutningskode betyr.
- Liveness, Readiness, and Startup Probes — hva hver probetype gjør, de fire sjekkmekanismene og alle konfigurasjonsfeltene med standardverdiene sine.
- Configure Liveness, Readiness and Startup Probes — gjennomgåtte eksempler, blant annet en startup-probe som gir et tregt program fem minutter.
- Health Check Response Format for HTTP APIs — det utløpte Internet-Draftet bak `application/health+json` og statusene pass, fail og warn.
- Using nginx as HTTP load balancer — de passive helsesjekkene i nginx med åpen kildekode, som skjer i selve trafikken (in-band), og innstillingene `max_fails` og `fail_timeout`.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er en helsesjekk?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/health-check/