Engineer

Health check

en automatisk test af, om en tjeneste fungerer

Hvad det er

Et health check er en lille test, der gentages efter en fast plan og spørger en kørende tjeneste, om den kan gøre sit arbejde, sammen med en regel for, hvad der sker, når svaret er nej. Testen kan være en HTTP-forespørgsel til en særlig sti, en kommando, der køres inde i en container, eller et forsøg på at åbne en TCP-port. Det er reglen, der gør tjekket til mere end en måling: et fejlet health check kan tage en server ud af rotation, genstarte en container, stoppe en udrulning eller vække nogen.

Docker bygger ideen ind i selve imaget. Instruktionen `HEALTHCHECK` i en Dockerfile angiver en kommando, som Docker kører inde i containeren med et fast interval. Afslutningskode 0 betyder sund, 1 betyder usund, og 2 er reserveret og må ikke bruges. En container med et tjek starter i tilstanden `starting`, bliver `healthy`, så snart ét tjek går igennem, og skifter til `unhealthy` efter et fastsat antal fejl i træk. Standardværdierne er et tjek hvert 30. sekund, en timeout på 30 sekunder og tre forsøg; en kørsel, der overskrider sin timeout, tæller som en fejl, og dens proces bliver dræbt. En valgfri startperiode giver et langsomt program tid til at starte op, og fejl inden for den tæller ikke. En Dockerfile rummer én sådan instruktion, den sidste vinder, hvis der er flere, og `HEALTHCHECK NONE` slår et tjek fra, der er arvet fra basisimaget.

Kubernetes deler spørgsmålet op i tre prober, som hver har sin egen konsekvens. En startup probe spørger, om programmet er færdigt med at starte; indtil den lykkes, venter de to andre, og bliver den ved med at fejle, dræbes containeren og overlades til sin genstartspolitik. En liveness probe spørger, om containeren skal genstartes, og er beregnet til tilstande som en deadlock, hvor processen kører, men ikke kommer videre. En readiness probe spørger, om containeren skal have trafik. Når den fejler, genstartes containeren ikke; i stedet fjernes Pod'ens adresse fra endpoints for hver Service, der matcher den, og forespørgslerne går til de andre replikaer, indtil proben svarer rigtigt igen.

Hver probe bruger én af fire mekanismer. En `httpGet`-probe går igennem på enhver statuskode fra 200 til og med 399. En `tcpSocket`-probe går igennem, hvis porten tager imod en forbindelse, også én, der bliver lukket med det samme. En `grpc`-probe går igennem, når tjenestens health check i gRPC svarer `SERVING`. En `exec`-probe kører en kommando i containeren og går igennem på afslutningskode 0. Medmindre andet er angivet, prober Kubernetes hvert tiende sekund, venter ét sekund på svar og handler efter tre fejl i træk.

Load balancere tilføjer en anden slags. Tjekkene ovenfor er aktive: noget sender en testforespørgsel, uanset om der er brugere eller ej. Et passivt tjek holder i stedet øje med den rigtige trafik. Sådan arbejder open source-udgaven af nginx: når en forespørgsel til en upstream-server ender med en fejl, markerer nginx serveren som fejlet og undgår den et stykke tid, og med standardværdien 1 for `max_fails` er ét mislykket forsøg nok. Ifølge nginx' dokumentation hører application health checks til de betalte NGINX Plus-abonnementer. Et passivt tjek koster ikke ekstra, men det opdager en ødelagt server gennem en brugers mislykkede forespørgsel, og det siger intet om en server, der ikke får trafik.

Brug begrebet i en beslutning

Brug »Hvad er et health check?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Overvågning og alarmer, der siger til, mens fejlen stadig er lille, og tallene, der viser, hvor tit den bliver ved med at være det.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har bygget det første observerbarhedsdashboard — virksomhedens første overblik over sine egne systemer

Har bygget det første observerbarhedsdashboard Kontakt os →

Hvorfor det betyder noget

En kørende proces er ikke det samme som en tjeneste, der virker. Dockers reference nævner det typiske eksempel: en webserver, der sidder fast i en uendelig løkke og er holdt op med at tage imod forbindelser, mens processen stadig lever. Alt, hvad der kun holder øje med, om processen findes, som en systemd-unit med en genstartspolitik, ser ingen fejl. Kun et spørgsmål, der stilles gennem den samme dør, som brugerne bruger, kan se forskellen, og det er hele begrundelsen for et health check: det erstatter »kører den?« med »svarer den?«.

To beslutninger afgør, om et tjek er nyttigt eller farligt. Den første er, hvad tjekket tester, og det svarer til en definition af »virker«, skrevet i kode. En handler, der returnerer 200 uden at røre noget, beviser, at webframeworket er

oppe; en handler, der også spørger databasen, beviser, at databasen kan nås, og fejler, når den ikke kan. Den anden er, hvad der sker ved en fejl, og den afgør, hvor langt skaden rækker. At tage én replika ud af rotation er billigt og kan rulles tilbage. At genstarte den smider dens igangværende arbejde og dens varme caches væk, og fejler alle replikaer det samme tjek på samme tid, gør en genstart af dem alle en langsom afhængighed til et totalt nedbrud.

Kubernetes advarer netop mod det. Dokumentationen skriver, at liveness probes skal sættes op, så de virkelig viser en fejl, der ikke kan rettes på anden måde, og at en forkert implementering kan give kaskadefejl: containere, der genstartes under høj belastning, klientforespørgsler, der fejler, og mere arbejde til de pods, der er tilbage. Rådet, der følger, er at holde de to spørgsmål adskilt. Når et program er strengt afhængigt af bagvedliggende tjenester, skal dets liveness probe gå igennem, så længe programmet selv er sundt, mens dets readiness probe også tjekker hver nødvendig tjeneste bagved. Så undgår trafikken en Pod, der kun kan svare med fejl, og Pod'en bliver ikke dræbt for en fejl, der ligger et andet sted.

Health checks fodrer også overvågningen. Et endpoint, som en load balancer spørger, kan også spørges udefra, og den alarm, der går, når det fejler, er ofte det første tegn på en driftshændelse. Da Irakli Nadareishvili skrev Internet-Draftet Health Check Response Format for HTTP APIs, fandtes der ingen standard for, hvad sådan et endpoint returnerer, og de fleste programmer nøjedes med HTTP-statuskoden eller brugte deres eget format. Draftet foreslog et JSON-svar med medietypen `application/health+json` og et obligatorisk `status`-felt med værdien `pass`, `fail` eller `warn`, hvor `pass` og `warn` hører sammen med en statuskode i 2xx eller 3xx og `fail` med en i 4xx eller 5xx. Den sidste udgave, fra oktober 2021, udløb i april 2022, og IETF angiver, at den hverken er godkendt af IETF eller har formel status. Den er brugbar som fælles ordforråd, ikke som standard.

Sådan ser det ud i praksis

I en Dockerfile følger tjekket med det image, det tjekker. Her spørger programmets eget lille probeprogram en sti på localhost og afslutter 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 kører proben hvert 15. sekund og giver hver kørsel tre sekunder. Fejl i det første minut tæller ikke, medmindre et tjek allerede er gået igennem, og på Docker Engine 25.0 eller nyere kører tjekkene i det minut hvert femte sekund, som er standardintervallet i startperioden, så en hurtig start bliver opdaget hurtigt. Efter tre fejl i træk markeres containeren som `unhealthy`. De første 4096 byte af probens output gemmes til `docker inspect`, og hver statusændring udsender en `health_status`-hændelse. Referencen beskriver en status, ikke en handling: noget andet skal reagere på den.

I Kubernetes kan de tre prober få hver sin sti, så hver af dem stiller sit eget spørgsmål:

```
containers:
  - name: api
    ports:
      - containerPort: 8080
    startupProbe:
      httpGet: { path: /livez, port: 8080 }
      periodSeconds: 5
      failureThreshold: 24 # op til 2 minutter til at starte
    livenessProbe:
      httpGet: { path: /livez, port: 8080 }
      periodSeconds: 10
      timeoutSeconds: 2
      failureThreshold: 6 # cirka et minut før en genstart
    readinessProbe:
      httpGet: { path: /readyz, port: 8080 }
      periodSeconds: 5
      timeoutSeconds: 2
      failureThreshold: 2
```

`/livez` svarer kun for selve processen, mens `/readyz` også tjekker databasen og alt andet, en forespørgsel ikke kan undvære. Startup proben bruger liveness-stien, som Kubernetes' dokumentation anbefaler, og giver 24 × 5 sekunder, før containeren opgives. Når den er gået igennem, tager liveness over med en lang lunte og readiness med en kort: omkring ti sekunders fejl tager Pod'en ud af Servicen, og et minuts fejl genstarter den. Standardværdierne ville give tre fejl med ti sekunders mellemrum, og det er for lidt til et program, der skal bruge to minutter på at indlæse sine data. Dokumentationens egen tommelfingerregel er, at en container, der normalt er længere tid om at starte end den indledende forsinkelse plus failureThreshold × periodSeconds, skal have en startup probe.

Et health check kan også vendes på hovedet. I overvågning med dead man's switch, fra september 2026, tjekker en vært med 464 MB, der kører en forge og en webserver, sig selv hvert kvarter med jitter: et lille script måler den tilgængelige hukommelse og den ledige diskplads og pinger kun en ekstern tjeneste, når begge ligger over deres tærskelværdier. Tavsheden er alarmen. En vært, der er løbet tør for hukommelse, har mistet netværket eller ikke længere kan starte, giver det samme signal som en, der bare er usund, og derfor blev den løsning valgt frem for en agent, der rapporterer en status. Tjekket læser tilgængelig hukommelse og ikke fri hukommelse, fordi kernen fylder ledig hukommelse med cache, og seks variabelnavne, der sagde »free«, blev rettet, så ingen ville »reparere« en fungerende alarm til en, der står permanent i alarm. Et modtaget signal bekræfter, at scriptet kørte, og at dets tærskelkontroller for tilgængelig hukommelse og ledig diskplads blev bestået; det dokumenterer ikke værtens generelle tilstand eller, at sitet svarer. Overvågning kan afsløre advarselstegn, men tidspunktet afhænger af belastningen og de forhold, der observeres.

Et tjek ved opstart besvarer et helt tredje spørgsmål. I e-mail med failover og leveringslog, fra 2025, går mailen gennem tre udbydere med automatisk failover, og foran det hele sender et opstartstjek en rigtig besked gennem hele stakken, styret af et flag. I udviklingsmiljøet logger en fejl en advarsel, og programmet fortsætter; i staging og produktion er fejlen fatal, og processen stopper i stedet for at udrulle et build, der ikke kan sende mail. Siden opsummerer resultatet sådan, at en hel kategori af lydløse fejl gik fra »en bruger opdager det dage senere« til »udrulningen stopper«.

Hvor det går galt

Tjekket beviser kun, at processen svarer. En handler, der returnerer 200 uden at røre noget, forbliver grøn, mens databasen er utilgængelig, disken er fuld, eller hver eneste rigtige forespørgsel ender med en fejl. Lad readiness-tjekket afprøve det, en forespørgsel har brug for, på en billig måde, og tilføj et tjek udefra, der følger brugerens vej gennem DNS, TLS og load balanceren, for kubelet sender sine prober direkte til Pod'ens egen IP-adresse og ser intet af det.

Liveness afhænger af noget uden for containeren. Spørger liveness proben databasen, får en langsom database den til at fejle i alle replikaer på én gang, og Kubernetes genstarter dem alle, netop mens tjenesten i forvejen er presset. Lad liveness kun svare for selve processen, som Kubernetes' dokumentation råder til, og læg afhængighederne i readiness, hvor en fejl leder trafikken væk i stedet for at dræbe containere.

Tiderne passer ikke til programmet. Kubernetes venter som standard ét sekund på svar, så et endpoint, der bliver langsommere under belastning, fejler, mens programmet blot har travlt, og en container, der starter langsommere, end dens liveness probe tillader, bliver dræbt, før den er klar, igen og igen. Mål, hvor lang tid endpointet tager under belastning, og sæt timeouten over det, og giv langsomme programmer en startup probe, der er dimensioneret efter den langsomste reelle opstart.

En omdirigering går igennem. En HTTP-probe i Kubernetes regner enhver status fra 200 til 399 som en succes, så en sti, der omdirigerer til en loginside, ser sund ud. Peger omdirigeringen på en anden vært, følger kubelet den ikke, men registrerer proben som vellykket med en advarsel. Med HTTPS springer kubelet desuden certifikatkontrollen over, så et udløbet certifikat får aldrig en probe til at fejle. Peg proberne på en sti, der svarer 200 direkte og returnerer en 5xx, når noget er galt, og tjek certifikater udefra.

Endpointet fortæller for meget til for mange. Et detaljeret svar, der opregner hver afhængighed, version og værtsnavn, oplyser en angriber lige så godt som en load balancer, og Internet-Draftets afsnit om sikkerhed bemærker, at health check-endpoints kan have brug for autentificering og rollebaseret adgangskontrol. Hold det offentlige svar til en status, og vis kun detaljerne på en intern port eller til autentificerede klienter.

Tjekket koster mere, end det bør. Kubernetes advarer om, at `exec` -prober starter nye processer ved hver kørsel, hvilket på tætpakkede noder med korte intervaller kan give et mærkbart CPU-forbrug på noden, og en probe, der kører en tung forespørgsel med få sekunders mellemrum på hver replika, er en belastning i sig selv. Foretræk HTTP-, TCP- eller gRPC-prober, hold endpointet billigt, og gem resultatet af et dyrt afhængighedstjek i nogle sekunder i stedet for at gentage det ved hver probe.

Læs mere

- Dockerfile reference: HEALTHCHECK — instruktionen, dens indstillinger og standardværdier, og hvad hver afslutningskode betyder.
- Liveness, Readiness, and Startup Probes — hvad hver probetype gør, de fire tjekmekanismer og alle konfigurationsfelter med deres standardværdier.
- Configure Liveness, Readiness and Startup Probes — gennemgåede eksempler, blandt andet en startup probe, der giver et langsomt program fem minutter.

- Health Check Response Format for HTTP APIs — det udløbne Internet-Draft bag `application/health+json` og dets statusværdier pass, fail og warn.
- Using nginx as HTTP load balancer — de passive in-band-tjek i open source-udgaven af nginx og indstillingerne `max_fails` og `fail_timeout`.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring udrulningsforløbet, driftsrisiciene og dem, der har vagten. Vi kan afgrænse en ændring med kontroller, en vej tilbage og tydeligt ansvar.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er et health check?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

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

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/health-check/