Engineer

Høj tilgængelighed

evnen til at holde en tjeneste kørende, når en del svigter

Hvad det er

Høj tilgængelighed er den egenskab, at en tjeneste bliver ved med at svare, når en af de maskiner, diske, netforbindelser eller processer, den hviler på, svigter. Den bygges som regel som en klynge: flere værter, der hver især kan køre tjenesten, software, der holder øje med dem, og en procedure, der flytter arbejdet fra en vært, der er gået ned, til en rask. ClusterLabs-projektet, der vedligeholder open source-klyngestakken Pacemaker og Corosync, beskriver sådan en klynge som noget, der koordinerer opstart, overvågning og genopretning af tjenester på tværs af en række værter, »for at minimere nedetid og samtidig bevare dataenes integritet«.

Sidste halvdel af den sætning vejer lige så tungt som den første. En tjeneste, der hurtigt kommer op igen med forkerte data, er ikke blevet mere tilgængelig. Den er blevet farlig.

PostgreSQL-dokumentationen skelner mellem to grunde til at lade flere databaseservere arbejde sammen. Den ene er, at en anden server hurtigt kan tage over, hvis den primære svigter; det kalder dokumentationen høj tilgængelighed. Den anden er, at flere computere kan levere de samme data; det kalder den belastningsfordeling. De to deler ofte maskineri, men de svarer på forskellige spørgsmål: høj tilgængelighed handler om at overleve en fejl, mens belastningsfordeling og skalerbarhed handler om at kunne bære mere arbejde.

Tilgængelighed er i sig selv et tal. Det traditionelle mål er tidsbaseret: den andel af en periode, hvor tjenesten kunne bruges. Googles SRE-bog forklarer, at man for tjenester, der er spredt over hele verden, og hvor en del næsten altid kører et eller andet sted, i stedet måler andelen af forespørgsler, der bliver besvaret korrekt. Begge dele skrives som procenter og omtales i antal nitaller.

Alle højt tilgængelige løsninger er bygget af de samme få dele:

- **Redundans.** En reserve for hver del, hvis svigt ville stoppe tjenesten: en ekstra vært, en standby-database, en ekstra netvej. Pacemaker understøtter blandt andet aktiv/passiv-par og N+1-opstillinger, hvor flere aktive noder deler én reserve.
- **Opdagelse.** Noget skal opdage fejlen. Kapitlet om failover i PostgreSQL-dokumentationen beskriver et heartbeat mellem primær og standby, der løbende tjekker forbindelsen, og om den primære stadig fungerer.
- **Failover.** At flytte arbejdet: forfremme standby-serveren, flytte tjenestens IP-adresse, starte processen på en anden vært.
- **Fencing.** At sikre sig, at den svigtende node virkelig er stoppet, før dens arbejde starter et andet sted.
- **Genopbygning af reserven.** Efter en failover kører systemet uden redundans, indtil en ny reserve er på plads.

Høj tilgængelighed er hverken backup eller katastrofegendannelse. Den beskytter mod, at en enkelt del svigter, mens resten af installationen stadig kører. En backup beskytter mod tab eller ødelæggelse af data, også den ødelæggelse, som en klynge trofast kopierer til hver eneste node, og katastrofegendannelse er planen for at miste hele datacentret.

Brug begrebet i en beslutning

Brug »Hvad er høj tilgængelighed?« 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

Alle tjenester har dele, der vil svigte. Hardware går i stykker, operativsystemkerner går ned, en virtuel maskine slettes ved en fejl, en netværkspartition skærer en rask node af fra resten af klyngen. Kubernetes-dokumentationen kalder den slags ufrivillige afbrydelser og stiller dem op over for frivillige, som nogen har planlagt, for eksempel at tømme en node for at

reparere eller opgradere den. Uden redundans bliver hver eneste af dem et udfald, også de planlagte, for den eneste måde at patche en enkelt server på er at tage den ned.

Netop det sidste er ofte den første fordel, et team mærker. Kapitlet om failover i PostgreSQL-dokumentationen kalder det nyttigt at skifte jævnligt mellem primær og standby: så kan hver maskine tages ned efter tur til vedligeholdelse, og hvert skift tester, at failover-mekanismen virker, når der er brug for den.

Nitallene kan omregnes til tid. For et år på 365 dage (525.600 minutter) og en måned på 30 dage (43.200 minutter) giver regnestykket:

Tilgængelighed	Tilladt pr. år	Tilladt pr. måned på 30 dage
99 %	3,65 dage	7,2 timer
99,9 %	8,76 timer	43,2 minutter
99,95 %	4,38 timer	21,6 minutter
99,99 %	52,56 minutter	4,32 minutter
99,999 %	ca. 5,26 minutter	ca. 26 sekunder

Hvert ekstra nital skærer rammen ned til en tiendedel, og prisen stiger hurtigere end det. SRE-bogen advarer om, at ét skridt mere i pålidelighed kan koste hundrede gange så meget som det forrige, og at brugerne efter et vist punkt ikke kan mærke forskel: Den, der sidder med en telefon, som selv kun er 99 % pålidelig, opdager ikke, om tjenesten bag den kører med 99,99 % eller 99,999 %. Bogen behandler derfor et tilgængelighedsmål som både et gulv og et loft. En tjeneste, der er langt mere tilgængelig, end den behøver at være, har brugt penge og udviklingstid, som kunne have gjort mere gavn andre steder.

Derfor er målet en forretningsbeslutning, der træffes tjeneste for tjeneste. Et service level objective skriver tallet ned; høj tilgængelighed er det ingeniørarbejde, der gør tallet muligt at nå.

Sådan ser det ud i praksis

Tag en opdigtet bookingtjeneste med tre lag: et weblag, et applikationslag og en PostgreSQL-database, hver på sin egen vært. Alle tal herunder er opfundet til eksemplet. Hvis hvert lag er oppe 99,9 % af tiden, og tjenesten har brug for alle tre, skal deres tilgængelighed ganges sammen. At fordoble et lag virker den anden vej, fordi begge kopier skal være nede på samme tid:

```
Tre lag i serie, hvert 99,9 % tilgængeligt:
  0,999 × 0,999 × 0,999 ≈ 0,997      ca. 26 timer nede om året

Ét lag fordoblet, hvis de to kopier svigter uafhængigt af hinanden:
  1 − 0,001 × 0,001 = 0,999999       ca. et halvt minut om året
```

Den anden udregning hviler på ét ord: uafhængigt. To kopier på samme vært, i samme rack, bag samme switch eller i samme zone hos samme cloududbyder deler skæbne, og så holder regnestykket op med at gælde. Formlen forudsætter også, at skiftet fra den ene kopi til den anden sker øjeblikkeligt og altid lykkes, og det kan kun test vise.

I Kubernetes er det netop det, topology spread constraints er til. Dokumentationens eget eksempel er en arbejdsbyrde, der kan skrumpe til to pods: Dem vil man ikke have på samme node, hvor én fejl kan tage hele arbejdsbyrden offline. En regel i pod-skabelonen spreder replikaerne over zoner:

```
topologySpreadConstraints:
    - maxSkew: 1
      topologyKey: topology.kubernetes.io/zone
      whenUnsatisfiable: DoNotSchedule
      labelSelector:
          matchLabels:
              app: booking
```

Med `DoNotSchedule` lader scheduleren hellere en pod vente end at placere den, så zonerne kommer til at afvige med mere end én. Planlagt vedligeholdelse kræver en sikring mere. Et PodDisruptionBudget begrænser, hvor mange replikaer af en applikation der må være nede på samme tid på grund af frivillige afbrydelser. `kubectl drain` respekterer budgettet, fordi kommandoen beder Eviction API'et om at fjerne hver pod; sletter man pods direkte, går man uden om budgettet:

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
    name: booking-pdb
spec:
    maxUnavailable: 1
```

```
    selector:
        matchLabels:
            app: booking
```

Dokumentationen anbefaler `maxUnavailable` frem for `minAvailable`, fordi det automatisk følger med, når antallet af replikaer ændrer sig.

Databasen er det sværeste lag, fordi den rummer tilstand. PostgreSQL kan holde en hot standby, der besvarer skrivebeskyttede forespørgsler, eller en warm standby, som man ikke kan forbinde sig til, før den er forfremmet. At vælge mellem synkron og asynkron replikering er at vælge, hvad en failover må koste af data. Ved synkron replikering venter bekræftelsen af en commit på det konfigurerede antal synkrone standby-servere på det konfigurerede bekræftelsesniveau. Indstillingerne `synchronous_standby_names` og `synchronous_commit` fastlægger betingelserne. Ved failover skal der vælges en standby-server med de bekræftede data; en asynkron standby kan stadig være bagud. Prisen for det første er hastighed: Dokumentationen bemærker, at en fuldt synkron løsning over et langsomt netværk kan mere end halvere ydeevnen.

PostgreSQL afgør ikke selv, hvornår der skal skiftes. Ifølge dokumentationen leverer den ingen systemsoftware, der kan opdage en svigtende primær og give standby-serveren besked; den opgave hører, ligesom flytningen af IP-adressen, til ekstern klyngesoftware. Selve forfremmelsen er ét trin, `pg_ctl promote` eller funktionen `pg_promote()`. Det, der kommer bagefter, er den del, planerne glemmer. Når den gamle primære er væk, kører klyngen på én server, hvad dokumentationen kalder en degenereret tilstand, indtil en ny standby er bygget. Værktøjet `pg_rewind` kan gøre det hurtigere på store klynger.

Her på sitet viser casen om e-mail med failover og leveringslog samme mønster brugt på en ekstern afhængighed. Tre e-mailudbydere sidder bag én grænseflade, failover mellem dem sker automatisk i stedet for som en ændring i konfigurationen foretaget under pres, og et tjek ved opstart sender en rigtig besked, så staging og produktion nægter at starte uden en udbyder, der virker. Ifølge casen betyder en udbyders dårlige eftermiddag nu en forringet vej frem for et udfald, til gengæld for tre integrationer, der skal holdes i live. Hvor tilgængeligt et system skal være, er en beslutning for sig: Siden om serviceniveau forklarer, at tilgængeligheden skrives ind i aftalen for hvert system for sig, fordi en rapport, der dannes om natten, og en betalingstjeneste kræver forskellige tal.

Hvor det går galt

Redundans, der deler skæbne. To kopier på én vært, én strømforsyning, én switch eller én zone hos en cloududbyder svigter sammen. Følg hver reserve tilbage til det, den afhænger af, og spred replikaer over fejldomæner med vilje frem for ved et tilfælde.

Split brain. Når noderne i en klynge mister forbindelsen til hinanden, men bliver ved med at køre, kan hver side konkludere, at den anden er død, og selv starte tjenesten. Pacemaker-dokumentationen beskriver følgen: Den samme IP-adresse aktiveret på to værter gør det uforudsigeligt, hvilken vært de indgående pakker når, og for en database kan skaden være ødelagte eller divergerende data. Fencing, også kaldet STONITH (Shoot The Other Node In The Head), er værnet: En fence-enhed, for eksempel en strømafbryder, som klyngen styrer, kobler den svigtende node fra, før dens arbejde starter et andet sted. Dokumentationen slår fast, at en klynge ikke sikkert kan komme sig over en node, der ikke svarer, uden fencing, og at fence-enheden ikke må afhænge af klyngens eget netværk, for så bliver det netværk et enkelt fejlpunkt.

En failover, ingen har prøvet. En standby, der aldrig er blevet forfremmet, er et håb, ikke en plan. Skift roller efter en fast plan, som PostgreSQL-dokumentationen lægger op til, skriv proceduren ned, og øv den, som chaos engineering øver andre fejl.

At glemme den degenererede tilstand. Efter en failover er der ingen reserve. Hvis genopbygningen af standby-serveren er langsom eller manuel, bliver den næste fejl, hvor lille den end er, et udfald. Behandl genetableringen af redundansen som en del af driftshændelsen, ikke som en opgave til senere.

At tro, at asynkron replikering ikke mister noget. Den kan miste de sidste transaktioner, der blev gennemført før fejlen. Afgør for hver database, om det er acceptabelt, og gør afvejningen synlig for dem, der ejer dataene.

At forveksle replikering med backup. En standby kopierer en fejlagtig `DELETE` lige så trofast som en korrekt skrivning. Høj tilgængelighed holder tjenesten oppe; kun en backup lader et team vende tilbage til dataene, som de var.

To noder og ingen til at afgøre striden. Pacemaker-dokumentationen påpeger, at en node, der er faldet ud på grund af strømsvigt, ser præcis ud som en node, man har mistet netværksforbindelsen til. Med to medlemmer ser hver side netop det. Kubernetes kræver mindst tre kontrolplan-noder i sin stablede opstilling, og PostgreSQL-dokumentationen nævner en tredje witness-server, der kan forhindre nogle forkerte failovers, men advarer om, at den ekstra kompleksitet kun er umagen værd med omhyggelig opsætning og grundig test.

At betale for nitaller, ingen kan se. Hvert ekstra nital koster mere end det forrige, og brugerne når tjenesten gennem netværk og enheder, der er mindre pålidelige end målet. Vælg tallet ud fra, hvad brugerne og forretningen har brug for, og byg derefter.

Læs mere

- ClusterLabs — open source-klyngestakken til høj tilgængelighed med projektets definition af, hvad en klynge koordinerer, og hvorfor den har brug for fencing.
- High Availability, Load Balancing, and Replication — PostgreSQL-kapitlet om primære servere og standby-servere, om synkrone og asynkrone løsninger og om, hvad de koster.
- Failover — PostgreSQL om heartbeats, witness-servere, STONITH, forfremmelse og den degenererede tilstand.
- Fencing — Pacemaker Explained om split brain, fence-enheder samt fencing og kvorum.
- Embracing Risk — Googles SRE-bog om at måle tilgængelighed, om hvad hvert nital koster, og om at vælge et mål.
- Disruptions — frivillige og ufrivillige afbrydelser i Kubernetes, og hvordan et PodDisruptionBudget begrænser dem.

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 høj tilgængelighed?« 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/high-availability/