Engineer

Høy tilgjengelighet

evnen til å holde en tjeneste i gang når en del svikter

Hva det er

Høy tilgjengelighet er egenskapen som gjør at en tjeneste fortsetter å svare når en av maskinene, diskene, nettforbindelsene eller prosessene den hviler på, svikter. Den bygges som regel som en klynge: flere verter som hver for seg kan kjøre tjenesten, programvare som holder øye med dem, og en prosedyre som flytter arbeidet fra en vert som har gått ned, til en frisk. ClusterLabs-prosjektet, som vedlikeholder klyngestakken Pacemaker og Corosync med åpen kildekode, beskriver en slik klynge som noe som koordinerer oppstart, overvåking og gjenoppretting av tjenester på tvers av en rekke verter, «for å minimere nedetid og samtidig bevare dataenes integritet».

Siste halvdel av den setningen veier like tungt som den første. En tjeneste som raskt kommer opp igjen med feil data, er ikke blitt mer tilgjengelig. Den er blitt farlig.

PostgreSQL-dokumentasjonen skiller mellom to grunner til å la flere databaseservere arbeide sammen. Den ene er at en annen server raskt kan ta over hvis den primære svikter; det kaller dokumentasjonen høy tilgjengelighet. Den andre er at flere datamaskiner kan levere de samme dataene; det kaller den lastfordeling. De to deler ofte maskineri, men de svarer på ulike spørsmål: høy tilgjengelighet handler om å overleve en feil, mens lastfordeling og skalerbarhet handler om å kunne bære mer arbeid.

Tilgjengelighet er i seg selv et tall. Det tradisjonelle målet er tidsbasert: andelen av en periode da tjenesten kunne brukes. Googles SRE-bok forklarer at for tjenester som er spredt over hele verden, og der en del nesten alltid kjører et eller annet sted, måler man i stedet andelen forespørsler som blir besvart riktig. Begge skrives som prosenttall, og man omtaler dem etter hvor mange niere de har.

Alle høyt tilgjengelige løsninger er bygget av de samme få delene:

- **Redundans.** En reserve for hver del som ville stoppe tjenesten om den sviktet: en ekstra vert, en standby-database, en ekstra nettvei. Pacemaker støtter blant annet aktiv/passiv-par og N+1-oppsett, der flere aktive noder deler én reserve.
- **Oppdagelse.** Noe må merke at feilen har skjedd. Kapitlet om failover i PostgreSQL-dokumentasjonen beskriver et heartbeat mellom primær og standby som hele tiden sjekker forbindelsen og om den primære fortsatt fungerer.
- **Failover.** Å flytte arbeidet: forfremme standby-serveren, flytte tjenestens IP-adresse, starte prosessen på en annen vert.
- **Fencing.** Å sørge for at noden som har sviktet, virkelig har stoppet før arbeidet dens starter et annet sted.
- **Gjenoppbygging av reserven.** Etter en failover kjører systemet uten redundans til en ny reserve er på plass.

Høy tilgjengelighet er verken en sikkerhetskopi eller katastrofegjenoppretting. Den beskytter mot at én del svikter mens resten av installasjonen fortsatt kjører. En sikkerhetskopi beskytter mot at data går tapt eller blir ødelagt, også når en klynge trofast har kopiert ødeleggelsen til hver eneste node, og katastrofegjenoppretting er planen for å miste hele datasenteret.

Bruk begrepet i en beslutning

Bruk «Hva er høy tilgjengelighet?» 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

Alle tjenester har deler som kommer til å svikte. Maskinvare går i stykker, operativsystemkjerner krasjer, en virtuell maskin slettes ved en feil, en nettverkspartisjon skjærer en frisk node av fra resten av klyngen. Kubernetes-dokumentasjonen kaller slikt ufrivillige avbrudd og stiller dem opp mot frivillige avbrudd som noen har planlagt, for eksempel å tømme en node for å reparere

eller oppgradere den. Uten redundans blir hvert eneste av dem et driftsavbrudd, de planlagte også, for den eneste måten å patche en enkelt server på er å ta den ned.

Nettopp det siste er ofte den første fordelen et team merker. Kapitlet om failover i PostgreSQL-dokumentasjonen kaller det nyttig å bytte jevnlig mellom primær og standby: da kan hver maskin tas ned etter tur for vedlikehold, og hvert bytte tester at failover-mekanismen virker når det trengs.

Nierne kan regnes om til tid. For et år på 365 dager (525 600 minutter) og en måned på 30 dager (43 200 minutter) blir regnestykket slik:

Tilgjengelighet	Tillatt per år	Tillatt per måned på 30 dager
99 %	3,65 dager	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

Hver ekstra nier kutter rammen ned til en tidel, og prisen stiger raskere enn som så. SRE-boken advarer om at ett trinn til i pålitelighet kan koste hundre ganger så mye som det forrige, og at brukerne etter et visst punkt ikke merker forskjell: den som sitter med en telefon som selv bare er 99 % pålitelig, merker ikke om tjenesten bak kjører med 99,99 % eller 99,999 %. Boken behandler derfor et tilgjengelighetsmål som både et gulv og et tak. En tjeneste som er langt mer tilgjengelig enn den trenger å være, har brukt penger og utviklingstid som kunne gjort mer nytte andre steder.

Derfor er målet en forretningsbeslutning som tas tjeneste for tjeneste. Et tjenestenivåmål skriver tallet ned; høy tilgjengelighet er ingeniørarbeidet som gjør det mulig å nå det.

Slik ser det ut i praksis

Ta en oppdiktet bookingtjeneste med tre lag: et weblag, et applikasjonslag og en PostgreSQL-database, hver på sin egen vert. Alle tallene nedenfor er funnet på for eksemplets skyld. Hvis hvert lag er oppe 99,9 % av tiden og tjenesten trenger alle tre, må tilgjengelighetene deres ganges sammen. Å doble et lag virker motsatt vei, fordi begge kopiene må være nede samtidig:

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

Ett lag doblet, hvis de to kopiene svikter uavhengig av hverandre:
  1 − 0,001 × 0,001 = 0,999999       ca. et halvt minutt i året
```

Den andre utregningen hviler på ett ord: uavhengig. To kopier på samme vert, i samme rack, bak samme svitsj eller i samme sone hos samme skyleverandør deler skjebne, og da slutter regnestykket å gjelde. Formelen forutsetter også at byttet fra den ene kopien til den andre skjer øyeblikkelig og alltid lykkes, og det kan bare testing vise.

I Kubernetes er det nettopp dette topology spread constraints er til for. Dokumentasjonens eget eksempel er en arbeidslast som kan krympe til to pods: dem vil man ikke ha på samme node, der én feil kan ta hele arbeidslasten offline. En regel i pod-malen sprer replikaene over soner:

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

Med `DoNotSchedule` lar scheduleren heller en pod vente enn å plassere den slik at sonene kommer til å avvike med mer enn én. Planlagt vedlikehold krever en sikring til. Et PodDisruptionBudget begrenser hvor mange replikaer av en applikasjon som kan være nede samtidig på grunn av frivillige avbrudd. `kubectl drain` respekterer budsjettet, fordi kommandoen ber Eviction-API-et om å kaste ut hver pod; sletter man pods direkte, går man utenom budsjettet:

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

Dokumentasjonen anbefaler `maxUnavailable` framfor `minAvailable`, fordi det automatisk følger med når antallet replikaer endrer seg.

Databasen er det vanskeligste laget, fordi den har tilstand. PostgreSQL kan holde en hot standby, som svarer på skrivebeskyttede spørringer, eller en warm standby, som man ikke kan koble seg til før den er forfremmet. Å velge mellom synkron og asynkron replikering er å velge hva en failover kan koste av data. Ved synkron replikering venter bekreftelsen av en commit på det konfigurerte antallet synkrone standby-servere på det konfigurerte bekreftelsesnivået. Innstillingene `synchronous_standby_names` og `synchronous_commit` fastsetter betingelsene. Ved failover må det velges en standby-server som har de bekreftede dataene; en asynkron standby kan fortsatt ligge etter. Prisen for det første er hastighet: dokumentasjonen påpeker at en fullt synkron løsning over et tregt nettverk kan mer enn halvere ytelsen.

PostgreSQL avgjør ikke selv når det skal byttes. Ifølge dokumentasjonen leverer den ingen systemprogramvare som kan oppdage en primær som har sviktet, og gi standby-serveren beskjed; den oppgaven hører, i likhet med flyttingen av IP-adressen, til ekstern klyngeprogramvare. Selve forfremmelsen er ett trinn, `pg_ctl promote` eller funksjonen `pg_promote()`. Det som kommer etterpå, er den delen planene glemmer. Når den gamle primære er borte, kjører klyngen på én server, det dokumentasjonen kaller en degenerert tilstand, til en ny standby er bygget. Verktøyet `pg_rewind` kan gjøre det raskere på store klynger.

Her på nettstedet viser oppføringen om e-post med failover og leveringslogging samme mønster brukt på en ekstern avhengighet. Tre e-postleverandører står bak ett grensesnitt, failover mellom dem skjer automatisk og ikke som en konfigurasjonsendring gjort under press, og en oppstartskontroll sender en ekte melding, slik at staging og produksjon nekter å starte uten en leverandør som virker. Ifølge oppføringen ble en leverandør med en dårlig ettermiddag til degradert drift i stedet for et driftsavbrudd, mot at tre integrasjoner må holdes i gang. Hvor tilgjengelig et system må være, er en egen beslutning: siden om servicenivå forklarer at tilgjengeligheten skrives inn i avtalen for hvert system for seg, fordi en rapport som lages om natten, og en betalingstjeneste krever ulike tall.

Hvor det går galt

Redundans som deler skjebne. To kopier på én vert, én strømforsyning, én svitsj eller én sone hos en skyleverandør svikter sammen. Følg hver reserve tilbake til det den er avhengig av, og spre replikaene over feildomener med vilje heller enn på slump.

Split brain. Når nodene i en klynge mister kontakten med hverandre, men fortsetter å kjøre, kan hver side konkludere med at den andre er død, og starte tjenesten selv. Pacemaker-dokumentasjonen beskriver følgen: samme IP-adresse aktivert på to verter gjør det uforutsigbart hvilken vert de innkommende pakkene når, og for en database kan skaden bli ødelagte eller divergerende data. Fencing, også kalt STONITH (Shoot The Other Node In The Head), er vernet: en fence-enhet, for eksempel en strømbryter som klyngen styrer, kobler fra noden som har sviktet, før arbeidet dens starter et annet sted. Dokumentasjonen slår fast at en klynge ikke trygt kan komme seg etter en node som ikke svarer, uten fencing, og at fence-enheten ikke må være avhengig av klyngens eget nettverk, for da blir det nettverket et enkelt feilpunkt.

En failover ingen har prøvd. En standby som aldri er blitt forfremmet, er et håp, ikke en plan. Bytt roller med faste mellomrom, slik PostgreSQL-dokumentasjonen legger opp til, skriv ned prosedyren, og øv på den slik chaos engineering øver på andre feil.

Å glemme den degenererte tilstanden. Etter en failover finnes det ingen reserve. Hvis gjenoppbyggingen av standby-serveren er treg eller manuell, blir den neste feilen, hvor liten den enn er, et driftsavbrudd. Behandle gjenopprettingen av redundansen som en del av driftshendelsen, ikke som en oppgave til senere.

Å tro at asynkron replikering ikke mister noe. Den kan miste de siste transaksjonene som ble fullført før feilen. Avgjør for hver database om det er akseptabelt, og gjør avveiningen synlig for dem som eier dataene.

Å forveksle replikering med sikkerhetskopi. En standby kopierer en feilaktig `DELETE` like trofast som en riktig skriving. Høy tilgjengelighet holder tjenesten oppe; bare en sikkerhetskopi lar et team gå tilbake til dataene slik de var.

To noder og ingen til å avgjøre striden. Pacemaker-dokumentasjonen påpeker at en node som har falt ut på grunn av strømbrudd, ser nøyaktig ut som en node man har mistet nettverksforbindelsen til. Med to medlemmer ser hver side nettopp det. Kubernetes krever minst tre kontrollplannoder i sitt stablede oppsett, og PostgreSQL-dokumentasjonen nevner en tredje witness-server som kan hindre enkelte feilaktige failovers, men advarer om at den ekstra kompleksiteten bare er verdt det med omhyggelig oppsett og grundig testing.

Å betale for niere ingen kan se. Hver ekstra nier koster mer enn den forrige, og brukerne når tjenesten gjennom nettverk og enheter som er mindre pålitelige enn målet. Velg tallet ut fra hva brukerne og virksomheten trenger, og bygg deretter.

Les mer

- ClusterLabs — klyngestakken med åpen kildekode for høy tilgjengelighet, med prosjektets definisjon av hva en klynge koordinerer, og hvorfor den trenger fencing.
- High Availability, Load Balancing, and Replication — PostgreSQL-kapitlet om primærservere og standby-servere, om synkrone og asynkrone løsninger og om hva de koster.
- Failover — PostgreSQL om heartbeats, witness-servere, STONITH, forfremmelse og den degenererte tilstanden.
- Fencing — Pacemaker Explained om split brain, fence-enheter og om fencing og kvorum.
- Embracing Risk — Googles SRE-bok om å måle tilgjengelighet, om hva hver nier koster, og om å velge et mål.
- Disruptions — frivillige og ufrivillige avbrudd i Kubernetes, og hvordan et PodDisruptionBudget begrenser dem.

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 høy tilgjengelighet?» 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/high-availability/