Engineer

Hög tillgänglighet

förmågan att hålla en tjänst igång när en del fallerar

Vad det är

Hög tillgänglighet är den egenskap som gör att en tjänst fortsätter att svara när någon av maskinerna, diskarna, nätförbindelserna eller processerna bakom den slutar fungera. Den byggs oftast som ett kluster: flera värdar som var för sig kan köra tjänsten, mjukvara som håller koll på dem och en procedur som flyttar arbetet från en värddator som har gått ner till en frisk. ClusterLabs-projektet, som underhåller klusterstacken Pacemaker och Corosync med öppen källkod, beskriver ett sådant kluster som något som samordnar start, övervakning och återställning av tjänster över en uppsättning värdar, ”för att minimera driftstopp och samtidigt bevara dataintegriteten”.

Den andra halvan av den meningen väger lika tungt som den första. En tjänst som snabbt kommer tillbaka med fel data har inte blivit mer tillgänglig. Den har blivit farlig.

PostgreSQL-dokumentationen skiljer mellan två skäl att låta flera databasservrar arbeta tillsammans. Det ena är att en andra server snabbt ska kunna ta över om den primära fallerar, vilket dokumentationen kallar hög tillgänglighet. Det andra är att flera datorer ska kunna leverera samma data, vilket den kallar lastbalansering. De två delar ofta maskineri, men de löser olika problem: hög tillgänglighet handlar om att överleva ett fel, medan lastbalansering och skalbarhet handlar om att klara mer arbete.

Tillgänglighet är i sig ett tal. Det traditionella måttet är tidsbaserat: den andel av en period då tjänsten gick att använda. Googles SRE-bok förklarar att man för tjänster som är spridda över hela världen, där någon del nästan alltid är igång någonstans, i stället mäter andelen förfrågningar som besvaras korrekt. Båda skrivs som procenttal och anges i antal nior.

Alla högt tillgängliga lösningar byggs av samma fåtal delar:

- **Redundans.** En reserv för varje del vars bortfall skulle stoppa tjänsten: en extra värddator, en standby-databas, en extra nätväg. Pacemaker stöder bland annat aktiv/passiv-par och N+1-uppsättningar, där flera aktiva noder delar på en reserv.
- **Upptäckt.** Något måste märka att felet har inträffat. Kapitlet om failover i PostgreSQL-dokumentationen beskriver ett heartbeat mellan primär och standby som hela tiden kontrollerar förbindelsen och om den primära fortfarande fungerar.
- **Failover.** Att flytta arbetet: befordra standby-servern, flytta tjänstens IP-adress, starta processen på en annan värddator.
- **Fencing.** Att se till att noden som har fallerat verkligen har stannat innan dess arbete startar någon annanstans.
- **Återuppbyggnad av reserven.** Efter en failover körs systemet utan redundans tills en ny reserv är på plats.

Hög tillgänglighet är varken en säkerhetskopia eller katastrofåterställning. Den skyddar mot att en enskild del går sönder medan resten av anläggningen fortfarande är igång. En säkerhetskopia skyddar mot att data går förlorade eller förstörs, även när ett kluster troget har kopierat förstörelsen till varenda nod, och katastrofåterställning är planen för att förlora hela datacentret.

Använd begreppet i ett beslut

Använd ”Vad är hög tillgänglighet?” 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

Alla tjänster har delar som kommer att gå sönder. Hårdvara slutar fungera, operativsystemets kärna kraschar, en virtuell maskin raderas av misstag, en nätverkspartition skär av en frisk nod från resten av klustret. Kubernetes-dokumentationen kallar sådant ofrivilliga avbrott och ställer dem mot frivilliga avbrott som någon har planerat, till exempel att tömma en nod för att reparera

eller uppgradera den. Utan redundans blir vart och ett av dem ett driftstopp, även de planerade, eftersom det enda sättet att patcha en ensam server är att stänga av den.

Just det sista är ofta den första fördelen ett team märker. Kapitlet om failover i PostgreSQL-dokumentationen kallar det användbart att regelbundet växla mellan primär och standby: då kan varje maskin tas ner i tur och ordning för underhåll, och varje växling testar att failover-mekanismen fungerar när den behövs.

Niorna går att räkna om till tid. För ett år på 365 dagar (525 600 minuter) och en månad på 30 dagar (43 200 minuter) blir räkningen:

Tillgänglighet	Tillåtet per år	Tillåtet per månad på 30 dagar
99 %	3,65 dagar	7,2 timmar
99,9 %	8,76 timmar	43,2 minuter
99,95 %	4,38 timmar	21,6 minuter
99,99 %	52,56 minuter	4,32 minuter
99,999 %	ca 5,26 minuter	ca 26 sekunder

Varje extra nia skär ner utrymmet till en tiondel, och kostnaden stiger snabbare än så. SRE-boken varnar för att ett steg till i tillförlitlighet kan kosta hundra gånger så mycket som det förra, och för att användarna efter en viss punkt inte märker någon skillnad: den som sitter med en telefon som själv bara är 99 % tillförlitlig märker inte om tjänsten bakom kör med 99,99 % eller 99,999 %. Boken behandlar därför ett tillgänglighetsmål som både ett golv och ett tak. En tjänst som är mycket mer tillgänglig än den behöver vara har lagt pengar och utvecklingstid som hade gjort mer nytta någon annanstans.

Därför är målet ett affärsbeslut som fattas tjänst för tjänst. Ett servicenivåmål skriver ner siffran; hög tillgänglighet är ingenjörsarbetet som gör det möjligt att nå den.

Så ser det ut i praktiken

Ta en tänkt bokningstjänst med tre lager: ett webblager, ett applikationslager och en PostgreSQL-databas, var och en på sin egen värddator. Alla siffror nedan är påhittade för exemplets skull. Om varje lager är uppe 99,9 % av tiden och

tjänsten behöver alla tre, ska deras tillgänglighet multipliceras. Att dubblera ett lager verkar åt andra hållet, eftersom båda kopiorna måste vara nere samtidigt:

```
Tre lager i serie, vart och ett 99,9 % tillgängligt:
  0,999 × 0,999 × 0,999 ≈ 0,997      ca 26 timmar nere per år

Ett lager dubblerat, om de två kopiorna fallerar oberoende av varandra:
  1 − 0,001 × 0,001 = 0,999999       ca en halv minut per år
```

Den andra uträkningen vilar på ett ord: oberoende. Två kopior på samma värddator, i samma rack, bakom samma switch eller i samma zon hos samma molnleverantör delar öde, och då slutar räkningen att gälla. Formeln förutsätter också att bytet från den ena kopian till den andra sker omedelbart och alltid lyckas, och det kan bara tester visa.

I Kubernetes är det precis det som topology spread constraints finns till för. Dokumentationens eget exempel är en arbetslast som kan krympa till två poddar: dem vill man inte ha på samma nod, där ett enda fel kan slå ut hela arbetslasten. En regel i poddmallen sprider replikerna över zoner:

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

Med `DoNotSchedule` låter schemaläggaren hellre en podd vänta än att placera den där zonerna skulle skilja sig åt med mer än en. Planerat underhåll kräver ytterligare ett skydd. En PodDisruptionBudget begränsar hur många repliker av en applikation som får vara nere samtidigt på grund av frivilliga avbrott. `kubectl drain` respekterar budgeten, eftersom kommandot ber Eviction-API:et att avlägsna varje podd; den som raderar poddar direkt går runt budgeten:

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

```
    maxUnavailable: 1
    selector:
        matchLabels:
            app: booking
```

Dokumentationen rekommenderar `maxUnavailable` framför `minAvailable`, eftersom den inställningen automatiskt följer med när antalet repliker ändras.

Databasen är det svåraste lagret, eftersom den har tillstånd. PostgreSQL kan hålla en hot standby, som tar emot frågor som bara läser data, eller en warm standby, som man inte kan ansluta till förrän den har befordrats. Att välja mellan synkron och asynkron replikering är att välja vad en failover får kosta i data. Vid synkron replikering väntar bekräftelsen av en commit på det konfigurerade antalet synkrona standby-servrar på den konfigurerade bekräftelsenivån. Inställningarna `synchronous_standby_names` och `synchronous_commit` anger villkoren. Vid failover måste en standby-server med de bekräftade data väljas; en asynkron standby kan fortfarande ligga efter. Priset för det första är hastighet: dokumentationen påpekar att en helt synkron lösning över ett långsamt nätverk kan mer än halvera prestandan.

PostgreSQL avgör inte själv när en failover ska ske. Enligt dokumentationen levererar den ingen systemmjukvara som kan upptäcka en primär som har fallerat och meddela standby-servern; den uppgiften hör, liksom flytten av IP-adressen, till extern klustermjukvara. Själva befordran är ett enda steg, `pg_ctl promote` eller funktionen `pg_promote()`. Det som kommer efteråt är den del som planerna glömmer. När den gamla primära är borta körs klustret på en enda server, vilket dokumentationen kallar ett degenererat tillstånd, tills en ny standby har byggts. Verktyget `pg_rewind` kan snabba upp det på stora kluster.

Här på webbplatsen visar portföljposten om e-post med failover och leveransloggning samma mönster tillämpat på ett externt beroende. Tre e-postleverantörer sitter bakom ett gemensamt gränssnitt, failover mellan dem sker automatiskt i stället för som en konfigurationsändring under press, och en kontroll vid uppstart skickar ett verkligt meddelande, så att staging och produktion vägrar starta utan en leverantör som fungerar. Enligt portföljposten blev en leverantör som har en dålig eftermiddag ett nedsatt driftläge i stället för ett avbrott, till priset av tre integrationer att hålla vid liv. Hur tillgängligt ett system måste vara är ett beslut i sig: sidan om servicenivå förklarar att tillgängligheten skrivs in i avtalet för varje system för sig, eftersom en rapport som tas fram under natten och en betaltjänst behöver olika siffror.

Var det går fel

Redundans som delar öde. Två kopior på samma värddator, samma strömförsörjning, samma switch eller samma zon hos en molnleverantör går sönder tillsammans. Följ varje reserv bakåt till det den är beroende av, och sprid replikerna över feldomäner med avsikt i stället för av en slump.

Split brain. När noderna i ett kluster tappar kontakten med varandra men fortsätter att köra kan varje sida dra slutsatsen att den andra är död och själv starta tjänsten. Pacemaker-dokumentationen beskriver följden: samma IP-adress aktiverad på två värdar gör det oförutsägbart vilken värd de inkommande paketen når, och för en databas kan skadan bli förstörda eller divergerande data. Fencing, även kallat STONITH (Shoot The Other Node In The Head), är skyddet: en fence-enhet, till exempel en strömbrytare som klustret styr, kopplar bort noden som har fallerat innan dess arbete startar någon annanstans. Dokumentationen slår fast att ett kluster inte säkert kan återhämta sig från en nod som inte svarar utan fencing, och att fence-enheten inte får vara beroende av klustrets eget nätverk, eftersom det nätverket då blir en enskild felpunkt.

En failover som ingen har provat. En standby som aldrig har befordrats är ett hopp, inte en plan. Byt roller med jämna mellanrum, som PostgreSQL-dokumentationen föreslår, skriv ner proceduren och öva på den på samma sätt som chaos engineering övar på andra fel.

Att glömma det degenererade tillståndet. Efter en failover finns det ingen reserv. Om återuppbyggnaden av standby-servern är långsam eller manuell blir nästa fel, hur litet det än är, ett driftstopp. Behandla återställandet av redundansen som en del av incidenten, inte som en uppgift till senare.

Att tro att asynkron replikering inte förlorar något. Den kan förlora de sista transaktionerna som genomfördes före felet. Avgör för varje databas om det är acceptabelt, och gör avvägningen synlig för dem som äger data.

Att förväxla replikering med en säkerhetskopia. En standby kopierar en felaktig `DELETE` lika troget som en korrekt skrivning. Hög tillgänglighet håller tjänsten uppe; bara en säkerhetskopia låter ett team gå tillbaka till data så som de var.

Två noder och ingen skiljedomare. Pacemaker-dokumentationen påpekar att en nod som har fallit bort på grund av strömavbrott ser exakt likadan ut som en nod som man har tappat nätverkskontakten med. Med två medlemmar ser varje sida just det. Kubernetes kräver minst tre kontrollplansnoder i sin staplade uppsättning, och PostgreSQL-dokumentationen nämner en tredje witness-server som kan förhindra vissa felaktiga failovers, men varnar för att den extra komplexiteten bara lönar sig med noggrann konfiguration och grundlig testning.

Att betala för nior som ingen kan se. Varje extra nia kostar mer än den förra, och användarna når tjänsten via nätverk och enheter som är mindre tillförlitliga än målet. Välj siffran utifrån vad användarna och verksamheten behöver, och bygg sedan efter den.

Läs mer

- ClusterLabs — klusterstacken med öppen källkod för hög tillgänglighet, med projektets definition av vad ett kluster samordnar och varför det behöver fencing.
- High Availability, Load Balancing, and Replication — PostgreSQL-kapitlet om primära servrar och standby-servrar, om synkrona och asynkrona lösningar och om vad de kostar.
- Failover — PostgreSQL om heartbeats, witness-servrar, STONITH, befordran och det degenererade tillståndet.
- Fencing — Pacemaker Explained om split brain, fence-enheter samt fencing och kvorum.
- Embracing Risk — Googles SRE-bok om att mäta tillgänglighet, om vad varje nia kostar och om att välja ett mål.
- Disruptions — frivilliga och ofrivilliga avbrott i Kubernetes, och hur en PodDisruptionBudget begränsar dem.

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 hög tillgänglighet?” 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/high-availability/