Engineer

Prometheus

en database som skraper og lagrer metrikker

Hva det er

Prometheus er et system for overvåking med åpen kildekode, bygd rundt sin egen tidsseriedatabase. Med noen sekunders mellomrom spør den programmene den holder øye med, om de ferske metrikkene deres over HTTP, lagrer hver verdi med et tidsstempel og lar deg spørre i, tegne grafer av og varsle på det den har tatt vare på. Navnet brukes om selve serveren og, like ofte, om den lille familien av komponenter som har vokst fram rundt den.

Den ble bygd hos SoundCloud, der arbeidet startet i 2012, og den hører nå til Cloud Native Computing Foundation. Den kom inn i 2016 som stiftelsens andre prosjekt, etter Kubernetes, og fikk status som «graduated» i 2018. Prometheus 3.0, kunngjort i november 2024, var den første hovedversjonen på sju år. Koden er for det meste skrevet i Go og utgitt under Apache 2.0-lisensen.

Alt den lagrer, er en tidsserie, navngitt med et metrikknavn og et sett etiketter (labels): nøkkel–verdi-par som sier hvilken forekomst av noe som ble målt, som i `api_http_requests_total{method="POST", handler="/messages"}`. Hver måling i serien er et 64-bits flyttall, eller et innebygd histogram (native histogram), med et tidsstempel i millisekunder. En annen verdi for en hvilken som helst etikett gir en annen serie. Det er det som gjør dataene lette å skjære opp etter vert, rute eller statuskode, og det er også det som gjør dem lette å overbelaste.

Programmene legger ut tallene sine på et HTTP-endepunkt, som regel `/metrics`, enten gjennom et klientbibliotek som er kompilert inn i koden, eller gjennom en exporter: en liten adapter som oversetter for et system Prometheus ikke kan instrumentere direkte, f.eks. en Linux-vert eller en database. Serveren finner målene sine gjennom tjenesteoppdagelse (service discovery) eller en statisk liste, og skraper dem etter en fast plan. En batchjobb som er ferdig før neste skraping, kan i stedet dytte tallene til en egen Pushgateway. Varslingsreglene kjører på

serveren. Et driftsvarsel som utløses, går til Alertmanager, som fjerner duplikater, grupperer, legger på stillheter (silences) og ruter varslene videre til e-post, PagerDuty, OpsGenie og lignende mottakere.

Den veies som regel opp mot innsamlere som dytter data, i tradisjonen etter Graphite og StatsD, mot generelle tidsseriedatabaser som InfluxDB, og mot metrikktjenester som andre drifter for deg. Grafana er det vanlige ansiktet foran den. Prometheus lagrer tall, ikke enkelthendelser, så en logglinje eller en trace hører hjemme et annet sted. Prosjektet sier også rett ut at det prioriterer pålitelighet framfor fullstendighet: fakturering per forespørsel, der hver eneste telling må være eksakt, er en jobb for et annet system.

Bruk begrepet i en beslutning

Bruk «Hva er Prometheus?» 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

Beslutningen Prometheus står midt i, er om dere skal drifte deres egen database for metrikker eller leie en, og hvis dere drifter den selv, hvor stor den må være.

Å drifte den selv koster ingen lisens. Kostnaden er en maskin og timene til den som holder den frisk. En tjeneste som andre drifter, tar det arbeidet bort og erstatter det med en regning som vokser med dataene som sendes inn, og da blir hver ny etikett en linje på en faktura. For noen titalls maskiner er en egen server som regel den billigste og enkleste veien. Regnestykket snur når antallet serier kommer opp i millioner og teamet heller vil betale enn bemanne.

Dimensjoneringen er snillere enn de fleste tror. Prosjektets dokumentasjon for lagring regner med ett til to byte per måling i snitt. Tusen serier som skrapes hvert 15. sekund og beholdes i standardperioden på 15 dager, blir 86,4 millioner målinger, ca. 86–173 MB på disk. Det er sjelden disken som tar slutt, men minnet. Blokken som fylles akkurat nå, ligger i minnet, beskyttet mot krasj av en write-ahead-logg, og hver aktiv serie koster noe der, enten noen spør etter den eller ikke.

Hver server er selvstendig med vilje. Den lener seg ikke på lagring over nettverket, så den fortsetter å svare mens infrastrukturen den overvåker, faller sammen, og det er da den trengs mest. Prisen er at den lokale lagringen verken er klynget eller replikert. Høy tilgjengelighet betyr to like servere som skraper de samme målene, og historikk lenger enn noen uker betyr å sende en kopi med `remote_write` til et lager som er bygd for å ta vare på den.

Å hente i stedet for å ta imot gir også en praktisk gevinst. Fordi serveren er den som spør, merker den når et mål slutter å svare, og registrerer det som at `up` faller til null. Og en ingeniør kan åpne den samme `/metrics` -siden i en nettleser og se nøyaktig det serveren ser, så en uenighet mellom grafen og applikasjonen er rask å avgjøre.

Slik ser det ut i praksis

I grunnleggerens arbeid dukker Prometheus opp som innsamleren bak et dashbord: én gang på en flåte av databaseservere, og én gang som andre versjon av en visning hele kontoret fulgte med på.

Databasesaken var ti servere med PostgreSQL 14 og MS SQL Server 2019 på virtuelle servere med Ubuntu 20.04 LTS. Prometheus og Grafana fulgte CPU, minne, disk-I/O og spørringsytelse i sanntid, ved siden av automatiserte sikkerhetskopier, regler for brannmurer og egne skript for logganalyse. Oppetiden holdt seg på 99,9 % på alle de ti serverne. Databaseverter passer godt for skraping, fordi tallene som varsler problemer der, en disk som fylles eller en spørring som blir tregere, flytter seg over timer. En måling med noen sekunders mellomrom ser dem lenge før en bruker gjør det.

Kontorvisningen viser steget fra en hjemmelaget innsamler til Prometheus. I en tidligere rolle bygde grunnleggeren en organisasjons første dashbord for observerbarhet. Det begynte med å lese CPU, RAM, disk, temperatur og mer fra eksterne Linux-servere over SSH, og fulgte Docker-containere på samme måte. En andre versjon ble bygd på nytt med Prometheus og Grafana, rundt metrikkene som ble valgt sammen med DevOps- og utviklingsteamene: CPU-utnyttelse, minnebruk, oppetid for tjenester og API-latens, med varsler når terskelverdier ble overskredet. Med dashbordet på plass oppdaget og løste teamene hendelser 40 % raskere. Å hente tall over SSH er en fin start for en håndfull maskiner. Det som lar den samme visningen fortsette å vokse, er en skraper med etiketter og et spørrespråk.

Hvor det går galt

Minnet klatrer etter en utgivelse som ikke endret noe viktig. Noen la til en etikett med ubegrensede verdier: en bruker-ID, en e-postadresse, en full URL-sti, en forespørsels-ID. Hver nye verdi ble en ny serie, og serveren holder nå millioner der den holdt tusener. Navneveiledningen advarer mot nettopp dette. Hold etikettverdiene til små, kjente sett, send detaljer per forespørsel til logger eller traces, og finn de verste synderne med en spørring som denne:

```
topk(10, count by (__name__) ({__name__=~".+"}))
```

En graf viser et stup eller en topp som aldri skjedde. En teller går bare opp, og den faller til null når prosessen starter på nytt. Å tegne den rå verdien, eller trekke fra for hånd, gjør hver omstart til et krasj eller en bølge. `rate()` forstår nullstillinger, så ta raten først og summer etterpå, over et vindu som er flere skrapeintervaller bredt:

```
sum by (job) (rate(http_requests_total{status=~"5.."}[5m]))
```

En opptaksregel (recording rule) som summerer en teller før raten er tatt, gjemmer nullstillingene inne i summen og bringer de falske stupene tilbake.

En tjeneste går ned, og feilvarselet står på grønt. En regel som «feilraten er over en terskel» evalueres over de seriene som finnes. Når målet forsvinner, forsvinner seriene også, uttrykket gir et tomt resultat, og varselet oppheves automatisk akkurat når en driftshendelse er i gang. Varsle på `up == 0` for hvert mål som skrapes, bruk `absent()` for hver serie som alltid må finnes, og la noe utenfor serveren sjekke at serveren selv kjører.

Tallene fra forrige kvartal er borte når noen spør etter dem. Den lokale lagringen beholder 15 dager som standard og er én kopi på én disk. Den fungerer godt som et vindu mot det som nettopp har skjedd, og dårlig som arkiv. Øk oppbevaringstiden med omhu, send det som må vare, til langtidslagring gjennom `remote_write`, og hold tall som penger avhenger av, i en database som er fasiten.

Les mer

- Prometheus overview — prosjektets egen beskrivelse av hva det er, komponentene, og hvor det passer dårlig.
- Data model — metrikknavn, etiketter og målinger, og hvorfor hver etikettverdi starter en ny serie.
- Metric types — counter, gauge, histogram og summary, og hvilke spørsmål hver av dem kan svare på.
- Querying basics — selektorer i PromQL, intervallvektorer og typene et uttrykk kan returnere.
- Storage — blokker på to timer, write-ahead-loggen, oppbevaringstid og grensesnittet for `remote_write`.
- Metric and label naming — grunnenheter, endelsen `_total` og advarselen mot etiketter med høy kardinalitet.

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 Prometheus?» 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/prometheus/