Engineer

Prometheus

en database, der henter og gemmer metrikker

Hvad det er

Prometheus er et open source-system til overvågning, bygget op om sin egen tidsseriedatabase. Med få sekunders mellemrum spørger det de programmer, det holder øje med, om deres aktuelle tal over HTTP, gemmer hver værdi med et tidsstempel og lader dig forespørge, tegne grafer og sætte alarmer op på det, der er gemt. Navnet bruges både om selve serveren og om den lille familie af komponenter, der er vokset op omkring den.

Projektet blev bygget hos SoundCloud, hvor arbejdet begyndte i 2012, og hører i dag under Cloud Native Computing Foundation. Det blev optaget i 2016 som fondens andet projekt efter Kubernetes og nåede status som graduated i 2018. Prometheus 3.0, annonceret i november 2024, var den første hovedversion i syv år. Koden er overvejende skrevet i Go og udgivet under Apache 2.0-licensen.

Alt, hvad Prometheus gemmer, er tidsserier. En serie er navngivet af en metrik og et sæt labels, par af nøgle og værdi, der fortæller, hvilken forekomst af noget der blev målt, for eksempel `api_http_requests_total{method="POST", handler="/messages"}`. Hver måling i serien er et 64-bit flydende kommatal eller et native histogram, og den bærer et tidsstempel i millisekunder. Får en label en ny værdi, er det en ny serie. Det gør data nemme at skære op efter vært, rute eller statuskode, og det gør dem også nemme at overbelaste.

Programmerne viser deres tal frem på et HTTP-endpoint, som regel `/metrics`. Det sker gennem et klientbibliotek, der er bygget ind i koden, eller gennem en exporter: en lille adapter, der oversætter tallene fra et system, Prometheus ikke selv kan instrumentere, for eksempel en Linux-vært eller en database. Serveren finder sine mål via service discovery eller en statisk liste og henter, på engelsk scraper, dem efter en fast plan. Et batchjob, der slutter før næste hentning, kan i stedet skubbe sine tal til en separat push gateway. Alarmreglerne kører på

serveren, og de alarmer, der går af, sendes til Alertmanager, som fjerner dubletter, grupperer dem, håndterer silences og sender dem videre til e-mail, PagerDuty, OpsGenie og lignende modtagere.

Typisk sammenlignes Prometheus med push-baserede indsamlere i Graphite- og StatsD-traditionen, med generelle tidsseriedatabaser som InfluxDB og med hostede metriktjenester. Grafana er det sædvanlige ansigt udadtil. Prometheus gemmer tal og ikke hændelser, så en loglinje eller et trace hører hjemme et andet sted. Projektet siger også ligeud, at det prioriterer pålidelighed over fuldstændighed: fakturering pr. forespørgsel, hvor hver optælling skal være nøjagtig, er en opgave for et andet system.

Brug begrebet i en beslutning

Brug »Hvad er Prometheus?« 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

Beslutningen bag Prometheus handler om, hvorvidt man skal drive sin egen metrikdatabase eller leje en, og hvis man driver den selv, hvor stor den skal være.

Der er ingen licensbetaling. Udgiften er en maskine og timerne fra den, der holder den sund. En hostet tjeneste tager det arbejde væk og erstatter det med en regning, der vokser med mængden af data, man sender, så hver ny label bliver til en linje på fakturaen. Til et par dusin maskiner er den selvdrevne server som regel den billigste og enkleste vej. Regnestykket vender, når antallet af serier når op i millioner, og teamet hellere vil betale end bemande opgaven.

Størrelsesberegningen er venligere, end de fleste regner med. Projektets dokumentation om lagring sætter gennemsnittet til 1–2 byte pr. måling. Tusind serier hentet hvert 15. sekund og gemt i standardperioden på 15 dage giver 86,4 millioner målinger, cirka 86–173 MB på disken. Det er sjældent disken, der løber fuld. Det er hukommelsen: den blok, der er ved at blive fyldt, ligger i hukommelsen, beskyttet mod nedbrud af en write-ahead-log, og hver aktiv serie koster plads dér, uanset om nogen nogensinde spørger efter den.

Hver server er med vilje selvstændig. Den læner sig ikke op ad netværkslagring og bliver derfor ved med at svare, mens den infrastruktur, den overvåger, er ved at falde sammen, og det er netop dér, man har mest brug for den. Prisen er, at det lokale lager hverken er klyngebaseret eller replikeret. Høj tilgængelighed betyder to ens servere, der henter fra de samme mål, og historik ud over nogle få uger betyder en kopi sendt med `remote_write` til et lager, der er bygget til at gemme den.

At hente frem for at modtage giver også en praktisk gevinst. Fordi serveren selv spørger, opdager den, når et mål holder op med at svare, og registrerer det som `up` faldet til nul. Og en udvikler kan åbne den samme `/metrics`-side i en browser og se præcis det, serveren ser, så en uenighed mellem grafen og applikationen hurtigt bliver afgjort.

Sådan ser det ud i praksis

I grundlæggerens arbejde dukker Prometheus op som indsamleren bag et dashboard: én gang bag en gruppe databaseservere og én gang som anden version af et overblik, hele kontoret fulgte med i.

Databasesagen var ti servere med PostgreSQL 14 og MS SQL Server 2019 på virtuelle Ubuntu 20.04 LTS-servere. Prometheus og Grafana fulgte CPU, hukommelse, disk-I/O og forespørgselsydelse, mens det skete, side om side med automatiske backups, firewallregler og egne scripts til loganalyse. På tværs af alle ti servere lå oppetiden på 99,9 %. Databaseværter egner sig godt til at blive hentet fra, fordi de tal, der varsler problemer dér, en disk, der fyldes, eller en forespørgsel, der bliver langsommere, bevæger sig over timer, og en måling med få sekunders mellemrum ser dem længe før en bruger.

Kontoroverblikket viser skridtet fra en hjemmelavet indsamler til Prometheus. I en tidligere rolle byggede grundlæggeren en organisations første observability-dashboard. Det begyndte med at læse CPU, RAM, disk, temperatur og mere fra Linux-servere over SSH og holdt øje med Docker-containere på samme måde. En anden version blev bygget op på Prometheus og Grafana omkring de metrikker, der var valgt sammen med DevOps- og udviklingsteamene: CPU-udnyttelse, hukommelsesforbrug, oppetid for tjenester og API-latens, med alarmer, når en tærskel blev overskredet. Med dashboardet på plads opdagede og løste teamene hændelser 40 % hurtigere. Polling over SSH er en fin start for en håndfuld maskiner; en scraper med labels og et forespørgselssprog er det, der lader det samme overblik vokse videre.

Hvor det går galt

Hukommelsen stiger efter en release, der kun ændrede småting. Nogen har tilføjet en label, hvis værdier er ubegrænsede: et bruger-id, en e-mailadresse, en fuld URL-sti, et request-id. Hver ny værdi blev til en ny serie, og serveren holder nu på millioner, hvor den før holdt på tusinder. Navngivningsguiden advarer netop mod det. Hold labelværdier inden for små, kendte mængder, send detaljer pr. forespørgsel til logge eller traces, og find de værste syndere med en forespørgsel som denne:

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

En graf viser et fald eller en spids, der aldrig fandt sted. En tæller går kun op, og den falder til nul, når processen genstarter. Tegner man den rå værdi eller trækker værdierne fra hinanden i hånden, bliver hver genstart til et nedbrud eller en bølge af trafik. `rate()` forstår nulstillinger, så tag raten først og læg sammen bagefter, over et vindue på flere hentningsintervaller:

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

En recording rule, der summerer en tæller, før raten tages, gemmer nulstillingerne inde i summen og bringer de falske fald tilbage.

En tjeneste går ned, og dens fejlalarm forbliver grøn. En regel som en fejlrate over en tærskel evalueres over de serier, der findes. Når målet forsvinder, forsvinder dets serier også, udtrykket giver et tomt resultat, og alarmen ophører af sig selv på det værst tænkelige tidspunkt. Sæt en alarm på `up == 0` for hvert mål, brug `absent()` til serier, der altid skal være der, og lad noget uden for serveren tjekke, at serveren selv kører.

Tallene fra sidste kvartal er væk, når nogen spørger efter dem. Det lokale lager gemmer 15 dage som standard og er én kopi på én disk. Det fungerer som et vindue mod den seneste adfærd, og som arkiv skuffer det. Hæv opbevaringstiden med omtanke, send det, der skal holde, til langtidslagring via `remote_write`, og hold tal, som der hænger penge på, i en egentlig database.

Læs mere

- Prometheus overview — projektets egen beskrivelse af, hvad det er, hvilke komponenter det har, og hvornår det er det forkerte valg.
- Data model — metriknavne, labels og målinger, og hvorfor hver labelværdi starter en ny serie.
- Metric types — counter, gauge, histogram og summary, og hvilke spørgsmål hver af dem kan besvare.
- Querying basics — PromQL-selektorer, range vectors og de typer, et udtryk kan returnere.
- Storage — blokke på to timer, write-ahead-loggen, opbevaringstid og remote write.
- Metric and label naming — basisenheder, endelsen `_total` og advarslen mod labels med høj kardinalitet.

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 Prometheus?« 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/prometheus/