Engineer

Prometheus

en databas som hämtar in och lagrar mätvärden

Vad det är

Prometheus är ett system för övervakning med öppen källkod, byggt kring en egen tidsseriedatabas. Med några sekunders mellanrum ber det programmen som det bevakar om deras senaste värden över HTTP, lagrar varje värde med en tidsstämpel och låter dig ställa databasfrågor mot det, rita grafer och skicka larm utifrån det som har sparats. Namnet betecknar servern för sig och, lika ofta, den lilla familj av komponenter som har vuxit upp kring den.

Det byggdes på SoundCloud, där arbetet började 2012, och tillhör i dag Cloud Native Computing Foundation (CNCF). Projektet gick med 2016 som stiftelsens andra projekt, efter Kubernetes, och nådde 2018 den högsta mognadsnivån, graduated. Prometheus 3.0, som presenterades i november 2024, var den första större versionen på sju år. Koden är till största delen skriven i Go och publiceras under licensen Apache 2.0.

Allt som lagras är en tidsserie, namngiven av ett mätvärdesnamn och en uppsättning etiketter (labels): par av nyckel och värde som säger vilken instans av något som mättes, som i `api_http_requests_total{method="POST", handler="/messages"}`. Varje mätpunkt i serien är ett 64-bitars flyttal, eller ett inbyggt histogram (native histogram), med en tidsstämpel i millisekunder. Ett annat värde på vilken etikett som helst ger en annan serie. Det gör datan lätt att dela upp per värddator, sökväg eller statuskod, och det gör den också lätt att överbelasta.

Program exponerar sina systemmätvärden på en HTTP-ändpunkt, oftast `/metrics`, genom ett klientbibliotek som kompileras in i koden eller genom en exporterare (exporter): en liten adapter som översätter ett system som Prometheus inte kan instrumentera direkt, t.ex. en värddator med Linux eller en databas. Servern hittar sina mål genom tjänsteupptäckt (service discovery) eller

en statisk lista och hämtar data från dem med fasta intervall, det som kallas scraping. Ett batchjobb som avslutas före nästa hämtning kan i stället skicka sina värden till en separat push-gateway. Larmregler körs på servern, och utlösta larm går till Alertmanager, som tar bort dubbletter, grupperar dem, tillämpar tystningar och skickar dem vidare till e-post, PagerDuty, OpsGenie och liknande mottagare.

Prometheus vägs oftast mot push-baserade insamlare i traditionen från Graphite och StatsD, mot allmänna tidsseriedatabaser som InfluxDB och mot hanterade tjänster för mätvärden. Grafana är det vanliga ansiktet framför det. Prometheus lagrar tal snarare än händelser, så en loggrad hör hemma i loggningen och en spårning (trace) i ett system för spårning. Projektet säger också rakt ut att det prioriterar tillförlitlighet framför fullständighet: debitering per anrop, där varje räkning måste vara exakt, är ett jobb för ett annat system.

Använd begreppet i ett beslut

Använd ”Vad är Prometheus?” 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

Beslutet som Prometheus står mitt i är om du ska driva en egen databas för mätvärden eller hyra en, och om du driver den själv, hur stor den måste vara.

Att driva den själv kostar ingen licensavgift. Kostnaden är en maskin och timmarna hos den som håller den frisk. En hanterad tjänst tar bort det arbetet och ersätter det med en avgift som växer med mängden data som skickas dit, vilket gör varje ny etikett till en rad på en faktura. För några dussin maskiner är den egna servern oftast den billigare och enklare vägen. Kalkylen ändras när antalet serier når miljontals och teamet hellre betalar än bemannar.

Storleksräkningen är snällare än de flesta tror. Projektets dokumentation om lagring anger i genomsnitt en till två byte per mätpunkt. Tusen serier som hämtas var 15:e sekund och sparas under standardlagringstiden på 15 dagar blir 86,4 miljoner mätpunkter, ungefär 86–173 MB på disk. Det är sällan disken som

tar slut. Det är minnet: blocket som fylls just nu ligger i minnet, skyddat mot krascher av en write-ahead-logg, och varje aktiv serie kostar något där, oavsett om någon någonsin frågar efter den.

Varje server är medvetet självständig. Den är inte beroende av nätverkslagring, så den fortsätter att svara medan infrastrukturen den bevakar faller ihop, och det är just då den behövs som mest. Priset är att den lokala lagringen varken är klustrad eller replikerad. Hög tillgänglighet betyder två identiska servrar som hämtar från samma mål, och historik längre än några veckor betyder att en kopia skickas med `remote_write` till ett lager som är byggt för att behålla den.

Att hämta i stället för att ta emot ger också en praktisk vinst. Eftersom servern är den som frågar märker den när ett mål slutar svara och registrerar det som att `up` sjunker till noll. Och en ingenjör kan öppna samma `/metrics`-sida i en webbläsare och se exakt det som servern ser, vilket gör att en oenighet mellan grafen och applikationen snabbt går att reda ut.

Så ser det ut i praktiken

I grundarens arbete dyker Prometheus upp som insamlaren bakom en dashboard: en gång för en flotta av databasservrar och en gång som andra versionen av en vy som hela kontoret tittade på.

I databasfallet hämtade Prometheus mätvärden från tio databasservrar med två motorer, PostgreSQL 14 och MS SQL Server 2019, alla på virtuella servrar med Ubuntu 20.04 LTS. Tillsammans med Grafana följde den CPU, minne, disk-I/O och prestanda för databasfrågor i realtid, vid sidan av automatiserade säkerhetskopior, regler för brandväggen och egna skript för logganalys. På alla tio servrarna höll sig upptiden på 99,9 %. Databasvärdar passar bra för den här sortens hämtning, eftersom de värden som förutsäger problem där, en disk som fylls eller en databasfråga som blir långsammare, rör sig över timmar. En mätpunkt med några sekunders mellanrum ser dem långt innan en användare gör det.

Kontorsvyn visar steget från en hemmabyggd insamlare till Prometheus. I en tidigare roll byggde grundaren en organisations första dashboard för observerbarhet. Den började med att läsa CPU, RAM, disk, temperatur med mera från Linux-servrar på distans via SSH, och bevakade Docker-containrar på samma sätt. En andra version byggdes om på Prometheus och Grafana, kring de mått som valdes tillsammans med DevOps- och utvecklingsteamen: CPU-användning,

minnesanvändning, tjänsternas upptid och API-latens, med larm vid överskridna tröskelvärden. Med dashboarden på plats upptäckte och löste teamen incidenter 40 % snabbare. Att polla över SSH är en bra start för en handfull maskiner.
En insamlare med etiketter och ett frågespråk är det som låter samma vy fortsätta växa.

Var det går fel

Minnet klättrar efter en release som inte ändrade något viktigt. Någon lade till en etikett vars värden saknar gräns: ett användar-ID, en e-postadress, en fullständig URL-sökväg, ett anrops-ID. Varje nytt värde blev en ny serie, och servern håller nu miljontals serier där den förut höll tusentals.
Namngivningsguiden varnar för exakt det här. Håll etikettvärdena till små, kända mängder, skicka detaljer per anrop till loggar eller spårningar och hitta de värsta bovarna med en PromQL-fråga som den här:

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

En graf visar ett stup eller en topp som aldrig inträffade. En räknare (counter) går bara uppåt och faller till noll när processen startar om. Att rita upp det råa värdet, eller subtrahera för hand, gör varje omstart till en krasch eller en rusning. `rate()` förstår nollställningar, så beräkna `rate()` först och summera efteråt, över ett fönster som är flera hämtintervall brett:

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

En inspelningsregel (recording rule) som summerar en räknare innan `rate()` tas gömmer nollställningarna inne i summan och tar tillbaka de falska stupen.

En tjänst går ned och dess larm för felfrekvens förblir grönt. En regel som felfrekvens över ett tröskelvärde utvärderas över de serier som finns. När målet försvinner försvinner även dess serier, uttrycket ger ett tomt resultat och larmet upphör av sig självt i värsta tänkbara ögonblick. Larma på `up == 0` för varje mål som hämtas, använd `absent()` för varje serie som alltid måste finnas och låt något utanför servern kontrollera att servern själv är igång.

Förra kvartalets siffror är borta när någon frågar efter dem. Den lokala lagringen sparar 15 dagar som standard och är en enda kopia på en enda disk. Den fungerar som ett fönster mot det senaste beteendet, och som arkiv sviker den. Höj lagringstiden med försiktighet, skicka det som måste bestå till långtidslagring via `remote_write` och håll siffror som pengar hänger på i en databas som är den auktoritativa källan för dem.

Läs mer

- Prometheus overview — projektets egen beskrivning av vad det är, vilka komponenter det har och var det passar dåligt.
- Data model — mätvärdesnamn, etiketter och mätpunkter, och varför varje nytt etikettvärde startar en ny serie.
- Metric types — counter, gauge, histogram och summary, och vilka frågor var och en kan besvara.
- Querying basics — selektorer i PromQL, intervallvektorer och de typer ett uttryck kan returnera.
- Storage — block på två timmar, write-ahead-loggen, lagringstiden och gränssnittet för remote write.
- Metric and label naming — basenheter, suffixet `_total` och varningen för etiketter med hög kardinalitet.

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 Prometheus?” 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/prometheus/