Engineer

Prometheus

メトリクスを定期的に取得して保存するデータベース

どのようなものか

Prometheusは、独自の時系列データベースを中心に据えたオープンソースの監視システムです。数秒ごとに監視対象のプログラムへHTTPで現在の数値を問い合わせ、各値をタイムスタンプとともに保存し、蓄積したデータの検索、グラフ化、アラート通知を可能にします。Prometheusという名前はサーバー単体を指すことも、その周辺に育った小規模なコンポーネント群全体を指すこともあり、どちらもよく使われます。

SoundCloudで2012年に開発が始まり、現在はCloud Native Computing Foundationに属しています。2016年にKubernetesに次ぐ同財団2番目のプロジェクトとして受け入れられ、2018年に卒業プロジェクトとして認定されました。2024年11月に発表されたPrometheus 3.0は、7年ぶりのメジャーリリースでした。コードの大部分はGoで書かれ、Apache 2.0ライセンスで公開されています。

保存されるものはすべて時系列で、メトリクス名とラベルの組み合わせで識別されます。ラベルは、計測対象のどのインスタンスを測ったかを示すキーと値の組で、たとえば `api_http_requests_total{method="POST", handler="/messages"}` のように指定します。その時系列の各サンプルは、64ビット浮動小数点値またはネイティブヒストグラムに、ミリ秒単位のタイムスタンプを添えたものです。どれか1つのラベルの値が違えば、別の時系列です。この仕組みによりホスト、ルート、ステータスコードなどでデータを切り分けやすくなりますが、同時に過負荷も招きやすくなります。

プログラムは、コードに組み込むクライアントライブラリか、エクスポーターを通じて、通常は `/metrics` というHTTPエンドポイントで数値を公開します。エクスポーターは、Linuxホストやデータベースなど、Prometheusが直接計測機能を組み込めないシステムの情報を変換する小さなアダプターです。サーバーはサービスディスカバリーか固定リストで対象を見つけ、予定に従って読み取ります。次の読み取り前に終了してしまうバッチジョブは、代わりに独立したプッシュゲ

ートウェイへデータを送れます。アラートルールはサーバーで評価され、発火したアラートはAlertmanagerへ送られます。Alertmanagerは重複を除き、グループにまとめ、通知の抑止を適用し、メール、PagerDuty、OpsGenieなどの通知先へ振り分けます。

比較対象になるのは、GraphiteやStatsDの流れをくむプッシュ型の収集ツール、InfluxDBのような汎用時系列データベース、ホステッド型のメトリクスサービスなどです。表示画面には通常Grafanaが使われます。保存するのはイベントではなく数値なので、ログ行やトレースは別の場所に置きます。プロジェクト自身も、完全性より信頼性を優先すると明言しています。すべての件数が正確でなければならないリクエスト単位の課金は、別のシステムに任せる仕事です。

概念を判断に生かす

「Prometheusとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 障害が小さいうちに知らせる監視とアラートを整え、安定した状態をどれだけ維持できているかを数値で示します。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 社内初のオブザーバビリティダッシュボードを構築 — 社内システムの状態を初めて可視化

社内初のオブザーバビリティダッシュボードを構築 お問い合わせ →

なぜ重要なのか

Prometheusを選ぶ場面で問われるのは、メトリクス用データベースを自分たちで運用するかサービスを借りるか、そして自分たちで運用するならどの程度の規模が必要かです。

運用にライセンス料はかかりません。費用になるのはマシンと、正常に動かし続ける担当者の時間です。ホステッドサービスはその作業を引き受ける代わりに、送ったデータ量に応じた料金を請求します。新しいラベルを追加するたびに、請求項目も増えることになります。数十台程度なら、自前のサーバーの方が安く、簡単なことが一般的です。時系列が数百万に達し、担当者を置くより料金を払いたいとチームが考える段階では、その計算が変わります。

容量を計算すると、必要量は多くの人が想像するより小さく済みます。プロジェクトのストレージ文書では、平均して1サンプルあたり1〜2バイトとしています。1,000本の時系列を15秒ごとに読み取り、標準の保存期間である15日間保持すると、86,400,000サンプル、ディスク上ではおよそ86〜173 MBになります。

不足しやすいのはディスクではなくメモリです。書き込み中のブロックはメモリに置かれ、クラッシュへの備えとして先行書き込みログで保護されます。アクティブな時系列は、誰かが検索するかどうかにかかわらず、メモリを消費します。

各サーバーは、意図的に自己完結する設計です。ネットワークストレージに頼らないため、監視しているインフラが崩れかける、まさに最も必要なときにも応答を続けられます。その代償として、ローカルストレージはクラスタ化も複製もされません。高可用性を確保するには同じ対象を読み取る同一構成のサーバーが2台必要で、数週間を超える履歴が必要なら、長期保存向けのストレージへ `remote_write` でコピーを送ります。

送られてくるデータを受け取るのではなく、自分から取りに行くことにも実用上の利点があります。問い合わせるのはサーバーなので、対象が応答しなくなると気づき、 `up` がゼロに下がったこととして記録できます。また、エンジニアはブラウザーで同じ `/metrics` ページを開き、サーバーが見ているものをそのまま確認できます。そのため、グラフとアプリケーションの表示の食い違いをすばやく検証できます。

実際の活用例

創業者の仕事では、ダッシュボードの背後でデータを集める役目としてPrometheusが登場します。1つはデータベースサーバー群の監視、もう1つはオフィス全体が見ていた画面の第2版です。

データベースの事例では、PostgreSQL 14とMS SQL Server 2019を動かす10台のUbuntu 20.04 LTS仮想サーバーが対象でした。PrometheusとGrafanaでCPU、メモリ、ディスクI/O、クエリ性能をリアルタイムに追い、自動バックアップ、ファイアウォールルール、独自のログ分析スクリプトも併用しました。10台全体の稼働率は99.9%を維持しました。データベースホストは定期的な読み取りに向いています。ディスクが埋まる、クエリが遅くなるといった問題の前兆となる数値は数時間かけて変わるため、数秒ごとのサンプルで利用者が気づくよりずっと前に捉えられるからです。

オフィスの画面は、自作の収集ツールからPrometheusへ進んだ過程を示しています。創業者は以前の職務で、ある組織で初となるオブザーバビリティのダッシュボードを構築しました。最初は離れたLinuxサーバーからSSHでCPU、RAM、ディスク、温度などを読み取り、Dockerコンテナーも同じ方法で監視していました。第2版ではPrometheusとGrafanaで作り直し、DevOpsチームとエンジニアリングチームで選んだCPU使用率、メモリ使用量、サービス稼働時間、APIのレイテンシーを表示し、しきい値を超えたときに通知しました。ダッシュボードの導

入後、チームによるインシデントの検知と解決は40%速くなりました。数台のマシンならSSHでのポーリングはよい出発点です。同じ画面の対象を増やし続けられるのは、ラベルとクエリ言語を備えた収集ツールを使うからです。

問題が起きるところ

重要な変更のないリリース後にメモリ使用量が増える。 誰かがユーザーID、メールアドレス、完全なURLパス、リクエストIDなど、値の種類に上限のないラベルを追加しています。新しい値が現れるたびに時系列が増え、数千だったものが今では数百万になっています。命名ガイドは、まさにこの使い方を警告しています。ラベルの値は事前に分かる小さな集合に限定し、リクエストごとの詳細はログやトレースへ送り、次のようなクエリで時系列を最も増やしているメトリクスを探します。

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

起きてもいない急落や急増がグラフに出る。 カウンターは増える一方ですが、プロセスが再起動するとゼロに戻ります。そのままの値をグラフにしたり、手作業で差を取ったりすると、再起動のたびに急落や急増として現れます。`rate()` はリセットを考慮するため、読み取り間隔の数倍にあたる期間について、先に変化率を求めてから合計します。

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

変化率を求める前にカウンターを合計する記録ルールは、合計の中にリセットを隠してしまい、見かけだけの急落を再び生みます。

サービスが停止してもエラーのアラートは正常表示のまま。 エラー率がしきい値を超えた、といったルールは、存在する時系列に対して評価されます。対象が消えると時系列も消え、式は空の結果を返します。最も困る瞬間に、アラートが自動的に解消されてしまうのです。すべての読み取り対象について `up == 0` で通知し、常に存在すべき時系列には `absent()` を使い、サーバー自身が動いているかどうかは外部から確認させます。

前の四半期の数値を求められたときには、もう残っていない。 ローカルストレージの標準保存期間は15日間で、1台のディスクに1つのコピーしかありません。最近の動きを見る窓としては役立ちますが、アーカイブとしては期待に応え

られません。保存期間は慎重に延ばし、残す必要のあるものは `remote_write` で長期ストレージへ送り、金銭に関わる数値は正本となるデータベースに保存します。

参考資料

- Prometheusの概要 — プロジェクト自身による、Prometheusの定義、構成要素、適さない用途の説明です。
- データモデル — メトリクス名、ラベル、サンプルと、ラベル値が変わるたびに別の時系列になる理由を説明します。
- メトリクスの種類 — カウンター、ゲージ、ヒストグラム、サマリーと、それぞれで答えられる問いを扱います。
- クエリの基本 — PromQLのセレクター、範囲ベクトル、式が返せる型を説明します。
- ストレージ — 2時間のブロック、先行書き込みログ、保存期間、remote-writeインターフェースを扱います。
- メトリクスとラベルの命名 — 基本単位、接尾辞 `_total` 、値の種類が膨大なラベルへの警告を説明します。



用語集

実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「Prometheusとは？」がお客様の組織にどう当てはまるかをご相談ください。
現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/prometheus/