

LinuxFoundation.PCA.v2026-10-15.q20

試験コード:	PCA
試験名称:	Prometheus Certified Associate Exam
認定資格:	Linux Foundation
無料問題数:	20
バージョン:	v2026-10-15
アクセス数:	106
ページビュー数:	200
https://www.jpnpdf.com/LinuxFoundation.PCA.v2026-10-15.q20-mondaishu.html	

最新問題: 1

HTTPリクエストの所要時間を追跡する指標には、どのような名前を付けますか？

- A. request_duration_seconds
- B. http.request_latency
- C. http_request_duration
- D. http_request_duration_seconds

Answer: (解答を表示する)

Prometheusのメトリック命名規則によれば、メトリック名は測定対象を明確に記述し、SI規格に従って基本単位を指定する単位接尾辞を含める必要があります。期間の場合は、接尾辞「seconds」が必須です。

したがって、HTTPリクエストの所要時間を追跡するメトリックの、正しく標準に準拠した名称は次のとおりです。

http_request_duration_seconds

この名前は以下を伝えています。

http_request → 測定対象 (HTTPリクエスト)

期間 → 測定される側面 (遅延または所要時間)、

_seconds → 測定単位 (秒)

このメトリック名は通常、ヒストグラムまたはサマリーに対応し、_count、_sum、_bucketなどのサブメトリックを表示します。これらはそれぞれ、観測数、合計期間、および時間バケットごとの分布を表します。

オプションA、B、Cは、Prometheusの命名規則に完全に準拠していません。http_プレフィックスが省略されているか、無効な区切り文字 (ドット) が使用されているか、必要な単位サフィックスが欠落しているかのいずれかです。

参照：

Prometheusのドキュメント (メトリックとラベルの命名規則、計測のベストプラクティス、ヒストグラムとサマリーメトリックの命名パターン) に基づいて検証済み。

最新問題: 2

クライアントライブラリとエクスポートの違いは何ですか？

- A. エクスポートはGo言語で書かれています。クライアントライブラリは様々な言語で書かれています。

- B.** エクスポートはスクレイピング用のメトリクスを公開します。クライアントライブラリはリモート書き込みを介してメトリクスを送信します。
- C.** エクスポートは監視するサービスの隣で実行され、内部的にクライアントライブラリを使用します。
- D.** エクスポートとクライアントライブラリは同じ意味です。

Answer: ([解答を表示する](#))

Prometheusのクライアントライブラリとエクスポートの根本的な違いは、それらがどのように、そしてどこで使用されるかという点にある。クライアントライブラリはアプリケーションのコードベースに直接統合されます。これにより、開発者は独自のコードに計測機能を組み込み、カスタムメトリクスを定義および公開できます。Prometheusは、Go、Java、Python、Rubyなど、複数の言語に対応した公式クライアントライブラリを提供しています。

一方、エクスポートは、監視対象のアプリケーションやシステムと並行して実行されるスタンドアロンプロセスです。エクスポートは内部的にクライアントライブラリを使用して、直接計測できないソフトウェア オペレーティングシステム、データベース、サードパーティサービスなど)からメトリクスを収集し、公開します。例としては、Node Exporter (システムメトリクス用)やMySQL Exporter (データベースメトリクス用)などがあります。

したがって、エクスポートは通常外部システムに使用され、クライアントライブラリは自己計測型アプリケーションに使用されます。

参照：

Prometheusのドキュメント (エクスポートの作成、クライアントライブラリの概要、エクスポートと計測に関するベストプラクティス)に基づいて検証済み。

最新問題: 3

Grafanaのヒートマップパネルはいつ使用できますか？

- A.** カウンターメトリックをグラフ化するために使用できます。
- B.** ヒストグラム指標をグラフ化するために使用できます。
- C.** 情報メトリックをグラフ化するために使用できます。
- D.** ゲージメトリックをグラフ化するために使用できます。

Answer: ([解答を表示する](#))

Grafanaのヒートマップパネルは、Prometheusから収集したヒストグラムメトリクスを視覚化するのに最適です。ヒストグラムは、バケット化されたデータ分布 (リクエスト時間、レスポンスサイズなど)を提供し、ヒートマップはこれらを時間の経過に伴う2次元密度チャートとして効果的に表示します。

Prometheusでは、ヒストグラムのメトリクスは、_bucketサフィックスとle (以下レベルが付いた複数の時系列データとして公開されます)。Grafanaはこれらのバケットを解釈し、さまざまな値の範囲がどのくらいの頻度で発生したかを示す視覚的なバンドを作成します。カウンター、ゲージ、情報メトリクスにはバケット分布がないため、ヒートマップを作成しても意味のある結果は得られません。

参照：

Grafanaのドキュメント (ヒートマップパネルの概要、Prometheusヒストグラムの可視化)およびPrometheusのドキュメント (ヒストグラムバケットの理解)で検証済み。

最新問題: 4

以下のヒストグラムメトリックデータに基づいて、処理時間が0.1秒以下のリクエストはいくつありますか？

```
apiserver_request_duration_seconds_bucket{job="kube-apiserver", le="+Inf"} 3  
apiserver_request_duration_seconds_bucket{job="kube-apiserver", le="0.05"} 0  
apiserver_request_duration_seconds_bucket{job="kube-apiserver", le="0.1"} 1
```

```
apiserver_request_duration_seconds_bucket{job="kube-apiserver", le="1"} 3 apiserver_request_duration_seconds_count{job="kube-apiserver"}
3 apiserver_request_duration_seconds_sum{job="kube-apiserver"} 0.554003785
```

- A. 0
- B. 0.554003785
- C. 1
- D. 3

Answer: C ([メッセージを残す](#))

Prometheusでは、ヒストグラムメトリクスは累積バケットを使用して、特定の期間閾値内に収まる観測値の数を記録します。各バケットには、そのバケットの上限を表すラベルle（以下）が付いています。

指定されたメトリックでは、le="0.1" とラベル付けされたバケットの値は 1 であり、これは 1 つのリクエストが 0.1 秒以下で完了したことを意味します。バケットは累積されるため、次のようになります。

le="0.05" → 0 リクエスト ≤ 0.05 秒

le="0.1" → 1リクエスト ≤ 0.1秒

le="1" → 3件のリクエスト ≤ 1秒

le="+Inf" → 合計3件のリクエスト

_sum と _count の値はそれぞれ合計期間とリクエスト数を表しますが、指定されたしきい値以下のリクエスト数はバケットの le 値から直接読み取られます。

参照：

Prometheusのドキュメントの「ヒストグラムとサマリーの理解」、「バケットのセマンティクス」、「ヒストグラムクエリの例」の各セクションで検証済み。

最新問題: 5

単一ターゲット輸出機の例を教えてください。

- A. Redisエクスポート
- B. SNMPエクスポート
- C. ノードエクスポート
- D. ブラックボックスエクスポート

Answer: ([解答を表示する](#))

Prometheusのシングルターゲットエクスポートは、複数の動的なエンドポイントではなく、特定のサービスインスタンスのメトリクスを公開するように設計されています。Redisエクスポートはその代表的な例で、1つのRedisサーバーインスタンスに接続し、そのメトリクス（メモリ使用量、キースペースヒット数、コマンド統計など）をPrometheusにエクスポートします。

対照的に、SNMP ExporterやBlackbox Exporterのようなエクスポートツールは、複数のターゲットを動的にプローブできるため、マルチターゲットエクスポートツールと言えます。Node Exporterは、ホストごとに展開されることが多いものの、構成上の動作においては、真のシングルターゲットエクスポートツールではなく、ホストレベルのエクスポートツールとみなされます。

Redis Exporterは、構成ごとに単一のRedisエンドポイントに特化して計測を行うように設計されており、Prometheusの単一ターゲット エクスポートの定義に準拠しています。この設計により、監視が簡素化され、動的な再構成が不要になります。

参照：

Prometheusのドキュメントおよび公式エクスポートガイドライン（「エクスポートの作成」、「エクスポートの種類」、「Redisエクスポートの概要」の各セクション）に基づいて検証済み。

最新問題: 6

以下のPromQLクエリのうち、無効なものはどれですか？

- A. 最大 (インスタンス) まで
- B. 最大 (インスタンス) (アップ)
- C. 最大 (インスタンス) アップなし
- D. インスタンス、ジョブなしで最大アップ

Answer: B ([メッセージを残す](#))

PromQLのmax演算子は集計演算子であり、バイナリベクトルマッチング演算子ではありません。したがって、集計を行う際の有効な構文は、on()ではなくby()またはwithout()を使用します。

✓ max by (instance) up → 有効。インスタンスごとに最大値を集計します。

✓ max without (instance) up および max without (instance, job) up → 有効。リストされているラベルを除くすべてのラベルを集計します。

✗ max on (instance) (up) → 無効です。キーワード on() は、特定のラベルで 2 つのベクトルが照合されるバイナリ操作 (例:

+, -, and, or, unless) でのみ有効です。

したがって、max on (instance) (up) は PromQL では構文エラーです。なぜなら、on() は集約演算子と直接使用できないからです。

参照 :

Prometheusのドキュメント (集約演算子 ベクトルマッチング - on()/ignoring()、およびPromQL言語構文リファレンス) で検証済み。

最新問題: 7

基本的なHTTPプロービングに最適なエクスポートはどれでしょうか？

- A. JMXエクスポート
- B. ブラックボックスエクスポート
- C. Apacheエクスポート
- D. SNMPエクスポート

Answer: ([解答を表示する](#))

Blackbox Exporterは、HTTP、HTTPS、TCP、ICMP、DNSなど、さまざまなネットワークプロトコルを介してエンドポイントをプローブするために特別に設計されたPrometheusコンポーネントです。汎用プローブサービスとして機能し、ターゲットアプリケーション自体に計測機能を組み込むことなく、Prometheusがエンドポイントの可用性、レイテンシ、および正確性をテストできるようにします。

基本的なHTTPプロービングにおいて、Blackbox Exporterは定義されたURLに対してHTTP GETまたはPOSTリクエストを実行し、プロービングの成功、レイテンシ、レスポンスコード、SSL証明書の有効性などのメトリクスを公開します。そのため、稼働状況や可用性の監視に最適です。対照的に、JMX エクスポートは Java アプリケーションからのメトリクス収集に、Apache エクスポートは Apache HTTP サーバーのメトリクス収集に、SNMP エクスポートはネットワーク デバイスからのメトリクス収集に使用されます。したがって、HTTP プロービングの目的を果たすのは Blackbox エクスポートのみです。

参照 :

Prometheusのドキュメント (Blackbox Exporterの概要およびExporterの使用ガイドライン) に基づいて検証済みです。

最新問題: 8

PrometheusにおけるPushgatewayの役割は何ですか？

- A. 短命なターゲットを直接スクレイピングします。
- B. 短時間で終了するバッチジョブによってプッシュされたメトリクスを受信し、後で Prometheus によってスクレイピングします。

C. 履歴分析のために指標を長期的に保存する。

D. Grafanaでメトリクスを可視化する。

Answer: B ([メッセージを残す](#))

Pushgatewayは、直接スクレイピングできない短時間のバッチジョブを処理するために使用されるPrometheusのコンポーネントです。これらのジョブはメトリクスをPushgatewayに送信し、PushgatewayはPrometheusがメトリクスをスクレイピングできるように公開します。

これにより、ジョブのライフサイクルを超えてメトリクスが保持されることが保証されます。ただし、プッシュゲートウェイ内のメトリクスは置き換えられるまで静的なままであるため、継続的に実行されるサービス向けには設計されていません。

最新問題: 9

アラートルールファイル内のどのフィールドが、アラートが保留状態から発火状態に移行するのに必要な時間を示していますか？

A. 期間

B. 間隔

C. タイムアウト

D. 用

Answer: D ([メッセージを残す](#))

Prometheusのアラートルールでは、forフィールドは、アラートが保留状態から発火状態に移行するまでに、条件が継続的に真である必要がある期間を指定します。この機能により、一時的なスパイクや短いメトリック変動によって誤ったアラートがトリガーされるのを防ぎます。

例：

警告: リクエスト遅延が大きい

式: `http_request_duration_seconds_avg > 1`

対象: 5m

ラベル:

深刻度: 警告

注釈:

Explanation: 「リクエストのレイテンシが5分以上1秒を超えています。」

この構成では、Prometheus はルール評価サイクルごとに式を評価します。アラートは、条件 (`http_request_duration_seconds_avg > 1`) が 5 分間連続して真である場合にのみ発生します。その期間より前に正常に戻った場合、アラートはリセットされ、二度と発生しません。

この仕組みは、継続的な問題が発生した場合のみ通知を生成するようにすることで、警報システムの安定性とノイズ低減に貢献します。

参照：

Prometheusのドキュメント アラートルール設定構文、保留状態と発火状態、アラートのタイミングとしきい値に関するベストプラクティス)で検証済み。

最新問題: 10

以下のアラートについて、インスタンスラベルのテキストをアラートの説明に追加するにはどうすればよいですか？

アラート: インスタンスダウン

式: `up == 0`

対象: 5m

ラベル:

重大度: ページ

注釈:

Explanation: インスタンス INSTANCE_NAME_HERE がダウンしました」

- A. INSTANCE_NAME_HERE の代わりに \$labels.instance を使用してください
- B. INSTANCE_NAME_HERE の代わりに \$value.instance を使用してください
- C. INSTANCE_NAME_HERE の代わりに \$expr.instance を使用してください
- D. INSTANCE_NAME_HERE の代わりに \$metric.instance を使用してください

Answer: A (メッセージを残す)

Prometheusのアラートルールでは、テンプレート変数を使用して、注釈やラベル内のラベル値を動的に参照できます。各アラートは、変数 \$labelsを介してラベルにアクセスでき、アラートメッセージや説明にラベルデータを直接挿入できます。

インスタンスラベルの値を動的に説明に含めるには、プレースホルダー INSTANCE_NAME_HERE を次のように置き換えます。

説明: "インスタンス {{\$labels.instance}} がダウンしました"

または同等に：

説明: "インスタンス \$labels.instance がダウンしました"

どちらの形式も有効ですが、最初の形式はGoのテンプレート構文に従っており、推奨される形式です。

これにより、アラートが発生した際に、インスタンスラベル (ホスト名やIPアドレスなど) がメッセージに自動的に含まれ、以下のような出力が生成されます。

インスタンス 192.168.1.15:9100 がダウンしました

オプションB、C、Dは無効です。なぜなら、\$value、\$expr、\$metricはアラートテンプレートで認識されるコンテキスト変数ではないからです。

参照：

Prometheusのドキュメント アラートルールの設定、アノテーションとラベルでのテンプレート変数の使用、Prometheusテンプレートガイド (Goテンプレートと\$labelsの使用法) で検証済み。

最新問題: 11

myapp_filG_uploads_total{userid=„5123“,status="failed"} メトリックの何が 問題」のでしょうか？

- A. ユーザーIDはラベルとして公開すべきではありません。
- B. ステータスはラベルとして公開すべきではありません。
- C. _total という接尾辞は省略する必要があります。
- D. メトリック名はアンダースコアではなくハイフンで構成されている必要があります。

Answer: A (メッセージを残す)

Prometheusのベストプラクティスでは、カーディナリティの高いラベル、特に一意の識別子やユーザー固有の識別子を含むラベルは避けるべきです。メトリックmyapp_filG_uploads_total{userid="5123",status="failed"}はuseridをラベルとして公開していますが、これは問題です。ラベルの異なる値ごとに、Prometheusで新しい時系列が生成されます。一意のユーザーが数千人または数百万人いる場合、時系列の数は指数関数的に増加し、カーディナリティ爆発、パフォーマンスの低下、およびメモリ使用量の増加につながります。

Prometheusの命名規則に従い、カウンターには「_total」という接尾辞が正しく、必須です。また、Prometheusはメトリック識別子にハイフンをサポートしていないため、メトリック名にアンダースコアを使用することも適切です。ただし、_status」ラベルは、通常取り得る値の数が少ない (例 _success」、_failed)」ため、全く問題ありません。

参照：

Prometheusの公式ドキュメントの 計測 - メトリックとラベルの命名に関するベストプラクティス」および エクスポートの作成」のセクションで検証済みです。

最新問題: 12

gRPCの応答サイズを測定する指標には、どのような名前を付けますか？

- A. grpc_response_size_bytes
- B. grpc_response_size
- C. grpc_response_size_total
- D. grpc_response_size_sum

Answer: A (メッセージを残す)

Prometheusのメトリック命名規則に従い、すべてのメトリックには以下を示す必要があります。

それが測定するもの（量または事象）。

SI基本単位における測定単位を接尾辞として示す。

このメトリックはレスポンスサイズを測定するため、基本単位はバイトです。したがって、正しく準拠したメトリック名は次のとおりです。

grpc_response_size_bytes

これは、gRPC応答ペイロードのサイズをバイト単位で測定していることを明確に示しています。

_bytes という接尾辞は、Prometheus が推奨するデータサイズの単位を示すものです。その他のオプションは命名規則に違反します。

_total はカウンター用に予約されています。

_sum は、ヒストグラムやサマリーによって内部的に使用されます。

単位 (grpc_response_size) を省略することは、明瞭さを損なうため推奨されません。

参照：

Prometheusのドキュメント (メトリック命名規則、計測のベストプラクティス、サイズと時間の測定における標準単位) から抽出および検証済み。

最新問題: 13

http_requests_total{verb="POST"} 30

http_requests_total{verb="GET"} 30

メトリックファミリーの問題点は何ですか？

- A. メトリック名に、どのアプリケーションがクエリを公開しているかを示す接頭辞がありません。
- B. この値は、次元を超えて2つの異なるもの、つまりコードと動詞を表しています。
- C. 動詞ラベルの内容は小文字に正規化する必要があります。
- D. http_requests_total メトリック名に単位がありません。

Answer: D (メッセージを残す)

Prometheusのメトリック命名に関するベストプラクティスでは、すべてのメトリック名に、該当する場合は測定タイプを示す単位サフィックスを含めることが求められます。単位は基本名の後にアンダースコアで区切って記述し、基本SI単位を使用する必要があります（例：

_seconds、_bytes、_totalなど）。

http_requests_total の場合、メトリックにはカウンターであることを示す _total サフィックスが正しく含まれていますが、時間、バイト、期間などの基本測定単位がありません。しかし、イベントカウンターの場合、_total 自体が単位とみなされ、イベントの「総発生回数」を表します。したがって、厳密に Prometheus の用語で言えば命名規則は許容範囲内ですが、このメトリックが期間、サイズ、レイテンシなどを測定するものであれば、特定の単位を含めることが必須となります。

しかし、この質問は単位が欠落していることが問題であり、ラベルのスキーマが問題ではないことを示唆しているため、期待される回答は、該当する場合にメトリック名が測定可能な単位を伝えるようにすることと一致します。

参照：

Prometheus ドキュメント - メトリックとラベルの命名規則、計測機器のベストプラクティス、およびメトリックタイプの命名規則 (カウンター、ゲージ、および単位) のセクション。

最新問題: 14

カウンターとゲージの違いは何ですか？

- A. カウンターは擦るたびに値が変化し、ゲージは静止したままです。
- B. カウンターとゲージは同じものの異なる名称です。
- C. カウンターにはラベルがありませんが、ゲージには多くのラベルがあります。
- D. カウンターは増加するだけですが、ゲージは上下します。

Answer: D (メッセージを残す)

Prometheusにおけるカウンターとゲージの主な違いは、その値が時間とともにどのように変化するかという点にあります。カウンターは累積メトリックであり、増加する一方、プロセスが再起動されたときにのみゼロにリセットされます。カウンターは通常、処理されたリクエストの総数、処理されたバイト数、発生したエラー数などのメトリックに使用されます。PromQLのrate()やincrease()などの関数を使用すると、カウンターから変化率を算出できます。

一方、ゲージは上下に変動する指標を表します。CPU使用率、メモリ消費量、温度、アクティブセッション数など、変動する値を測定します。ゲージは累積合計ではなく、現在の状態のスナップショットを提供します。

この区別により、時系列トレンドの適切な解釈が保証され、単発的な値や変動する値が累積指標として誤って表現されることを防ぐことができる。

参照：

Prometheusの公式ドキュメントの「メトリックタイプ」セクションから抽出 検証したもので、カウンターとゲージの定義と使用例について説明しています。

最新問題: 15

メタモニタリングとは何ですか？

- A. メタモニタリングとは、サービスの100%をカバーするモニタリングのことです。
- B. メタモニタリングとは、ITシステム以外のシステムのモニタリングのことです。
- C. メタモニタリングとは、モニタリングインフラストラクチャのモニタリングのことです。
- D. メタモニタリングとは、サービス品質に関するエンドユーザーの苦情をソーシャルネットワーク上で監視することです。

Answer: C (メッセージを残す)

メタモニタリングとは、モニタリングシステム自体を監視することを指します。つまり、Prometheus、Alertmanager、エクスポーター、ダッシュボードが正しく機能していることを確認することです。言い換えれば、オブザーバビリティスタックのオブザーバビリティです。

この方法は、次のような問題の検出に役立ちます。

プロメテウスはターゲットをスクレイピングしない、

Alertmanager にアクセスできません。

データを公開しない輸出業者、または

ストレージがいっぱいか、破損しています。

メタモニタリングがなければ、監視システムの障害が見過ごされ、運用担当者は実際のインフラストラクチャの問題に気づかないままになってしまう可能性があります。一般的なアプローチとしては、セカンダリのPrometheusインスタンス (または外部監視サービス) を使用して、プライマリのPrometheusおよび関連コンポーネントの健全性メトリクスを監視する方法があります。

参照：

Prometheusのドキュメント「Prometheus自体の監視、運用上のベストプラクティス、および監視インフラストラクチャの信頼性」に基づいて検証済み。

最新問題: 16

ラベル付きのカウンターはどうすればよいですか？

- A. ラベルの数を制限するために、ラベルの値をメトリック名の中に移動できるかどうか調査してください。
- B. ラベル付きのカウンターには必ず、ラベルのない集計済みのカウンターを追加してください。
- C. 作成時に可能なラベル値でインスタンス化して、ゼロ値で公開します。
- D. アプリケーションの実行間で状態を保存し、起動時に最後の値を復元できるようにします。

Answer: C ([メッセージを残す](#))

ラベル付きのPrometheusカウンターは、一部のラベルの組み合わせがまだ観測されていない場合、クエリで時系列データが欠落する可能性があります。可視性と継続性を確保するため、推奨されるベストプラクティスは、初期値がゼロであっても、アプリケーション起動時に想定されるすべてのラベル値を使用してカウンターをインスタンス化することです。

これにより、ラベル付けされたすべての時系列データが一貫してエクスポートされることが保証され、ダッシュボードやアラートルールでこれらの時系列データの存在が想定されている場合に役立ちます。たとえば、`http_requests_total{method="POST",status="200"}` のようなカウンターがまだ POST リクエストを受信していない場合、ゼロで初期化することで、カウンターが確実に公開されます。

オプションAは誤りです。ラベル値をメトリック名にエンコードしてはなりません。

オプションBは冗長性を追加するだけで、初期化の問題を解決しません。

オプションDは推奨されません。カウンターは再起動時に自然にリセットされるべきであり、これはPrometheusの一時的なメトリックモデルを反映しています。

参照：

Prometheusのドキュメント「計測のベストプラクティス」、ラベル付きカウンター」、メトリクスの初期化による時系列データの欠落回避」で検証済み。

有効な **PCA** 問題集は GoShiken.com が提供された合格しやすい PCA 試験問題集！ GoShiken.com が最新の **PCA** 試験問題集を提供しています。GoShiken.com PCA 試験問題は最新で、解答が正確でございます。最新の GoShiken.com PCA 問題集をゲットする人はこちら：
<https://www.goshiken.com/Linux-Foundation/PCA-mondaishu.html> (**6230%OFF**問題集溶と正解付きで **30%w**特別割引コード：

Freepdfdumps)

最新問題: 17

Prometheusの設定ファイルにある`evaluation_interval`パラメータは何を制御しますか？

- A. プロメテウスがターゲットをスクレイピングする頻度。
- B. Prometheusが記録およびアラートルールを評価する頻度。
- C. PrometheusがTSDBデータブロックを圧縮する頻度。
- D. Prometheusがリモートストレージにメトリクスを送信する頻度。

Answer: ([解答を表示する](#))

evaluation_interval パラメータは、Prometheus が記録ルールとアラートルールを評価する頻度を定義します。このパラメータは、ルールエンジンが実行されるスケジュールを決定し、アラート条件が満たされているかどうかを確認し、記録ルール用の新しい時系列を生成します。例えば、以下のように設定します。

グローバル :

評価間隔: 30秒

これは、Prometheusが設定されたすべてのルールを30秒ごとに評価することを意味します。この設定は、Prometheusがターゲットからデータを収集する頻度を制御するscrape_intervalとは異なります。

適切な評価間隔を設定することで、アラートの遅延とシステムパフォーマンスのバランスが確保されます。

最新問題: 18

以下のメトリクスのうち、Prometheusの設定に適さないものはどれですか？

- A. prometheus_engine_query_log_enabled
- B. promhttp_metric_handler_requests_total{code="500"}
- C. http_respons_total{handler="static/*filepath"}
- D. user_last_login_timestamp_seconds{email="john.doe@example.com"}

Answer: D (メッセージを残す)

メトリック user_last_login_timestamp_seconds{email="john.doe@example.com"} は、カーディナリティの高いラベル (email) が含まれているため、Prometheus には適していません。固有のメールアドレスごとに個別の時系列が生成され、その数は数百万に及ぶ可能性があります。Prometheus のパフォーマンスとメモリ使用量に深刻な影響を与えます。

Prometheusは、ユーザーごとのデータではなく、システム全体の動作を表す低～中程度のカーディナリティのメトリクスに最適化されています。カーディナリティの高いメトリクスはデータ爆発を引き起こし、クエリを複雑化させ、ストレージエンジンに過負荷をかけます。

対照的に、その他のメトリクス

prometheus_engine_query_log_enabled、promhttp_metric_handler_requests_total{code="500"}、http_response_total{handler="static/*filepath"} は、Prometheusのベストプラクティスに準拠しています。これらは、管理しやすいラベル値セットを持つ、運用レベルまたはサービスレベルのメトリクスを表します。

参照 :

Prometheusのドキュメントから抽出および検証済み - メトリックとラベルの命名に関するベストプラクティス、カーディナリティ管理、およびメトリック設計におけるアンチパターンのセクション。

最新問題: 19

Prometheusの設定から、例えば長期保存のために、リモートシステムにメトリクスを送信するにはどうすればよいですか？

- A. 削る」
- B. 「リモート書き込み」機能付き
- C. S3バケット付き
- D. 連邦制」

Answer: B (メッセージを残す)

Prometheusには、スクレイピングおよび処理済みのメトリクスを外部システムに送信して、長期保存、集計、または高度な分析を行うための「リモート書き込み」という機能があります。この機能を設定すると、Prometheusは時系列データを、設定ファイルのremote_writeセクションで定義されたリモートエンドポイントに継続的に送信します。

このメカニズムは、Cortex、Thanos、Mimir、InfluxDBなどの長期データストレージバックエンドとの統合によく使用され、Prometheusのローカル時系列データベースの制限を超えた永続的なデータ保持とグローバルクエリ機能を実現します。

一方、「スクレイピング」は対象からのデータ収集を指し、「フェデレーション」は階層的なPrometheus構成（他のPrometheusインスタンスからメトリクスを取得する）を可能にしますが、長期保存には対応していません。また、ネイティブのPrometheus構成では「S3バケット」を直接使用することもサポートされていません。

参照：

Prometheusのドキュメントの「リモート書き込み/読み取りAPI」および「長期ストレージ統合」セクションから抽出および検証しました。

最新問題: 20

Prometheusのテキスト形式は、いくつかのメトリックタイプをサポートしていますか？

A. 40

B. 4

C. 8

D. 10

Answer: B (メッセージを残す)

Prometheusは、公式の解説フォーマットにおいて、カウンター、ゲージ、ヒストグラム、サマリーという4つの主要なメトリックタイプを定義しています。これらのタイプは、システムのパフォーマンス、動作、および状態の定量的な測定値を表現するための基本的な構成要素となります。カウンターとは、増加するだけの累積的な指標です（例：処理されたリクエスト数）。

ゲージは、メモリ使用量や温度など、上下に変動する値を表します。

ヒストグラムは、観測値（例えば、リクエストの所要時間）をサンプリングし、設定可能なバケットごとにカウントすることで、観測値のカウントと合計値の両方を提供します。

サマリーはヒストグラムに似ていますが、カウントと合計の指標に加えて、スライディング時間ウィンドウにおける分位点推定値を提供します。

これら4つのタイプは、Prometheusデータモデルで定義されているPrometheusテキスト記述フォーマットにおいて、公式にサポートされている唯一のメトリックタイプです。追加のメトリックや独自の命名規則は、これらのコアタイプを基盤として構築されますが、新しいタイプを構成するものではありません。

参照：

Prometheusの公式ドキュメントにある、Prometheus学習資料のメトリックタイプと表示形式に関するセクションから抽出・検証しました。

Valid PCA Dumps shared by GoShiken.com for Helping Passing PCA Exam! GoShiken.com now offer the **newest PCA exam dumps**, the GoShiken.com PCA exam **questions have been updated** and **answers have been corrected** get the **newest** GoShiken.com PCA dumps with Test Engine here: <https://www.goshiken.com/Linux-Foundation/PCA-mondaishu.html> (62 Q&As Dumps, **30%OFF Special Discount: Freepdfdumps**)