VictoriaMetrics Metrics 查詢

以繁體中文詳細說明:我需要查到盡可能多的VictoriaMetrics component metrics但查不到

當你想要查詢 **VictoriaMetrics** 所有 component（例如 `vminsert`、`vmstorage`、`vmselect`、`vmagent` 等）暴露出來的 **metrics** 時，確實會發現官網文件比較簡略，沒有像 Prometheus 或其他系統那樣完整的 metrics 列表。不過，我可以幫你整理幾種能「查到最多 metrics」的方法，搭配說明每個方法的細節與優缺點。

✅ 方法一：直接查詢 `/metrics` endpoint（最直接有效）

VictoriaMetrics 的所有 component 都遵守 Prometheus 標準格式，因此每個元件都會有一個 `/metrics` HTTP endpoint，可以直接取得目前可觀測的所有 metrics。

範例：

component	預設 port	範例查詢 URL
vminsert	`8480`	`http://<host>:8480/metrics`
vmselect	`8481`	`http://<host>:8481/metrics`
vmstorage	`8482`	`http://<host>:8482/metrics`
vmagent	`8429`	`http://<host>:8429/metrics`

🔍 你可以打開瀏覽器或用 curl 指令查看：

```
curl http://<你的 vmagent IP>:8429/metrics
```

💡 這個頁面會顯示出目前所有 metrics 名稱、labels 和數值，例如：

```
vm_rows_inserted_total{type="gauge"} 123456
vm_ingest_ingested_samples_total 987654
```

✅ 方法二：Prometheus 抓取後用 Label Filter 檢視

如果你有部署 Prometheus 來抓取 VictoriaMetrics 指標，可以透過 Prometheus 的「Explore」功能來查看有哪些 metrics。

1. 開啟 Prometheus Web UI（預設 `**`http://<prometheus>:9090`**`）
2. 輸入 `**`vm`**` 開頭或其他字首的關鍵字（如 `**`vm_`**`）
3. 查看 auto-complete 裡出現的所有 metrics 名稱

這種方法不需要自己查 endpoint，但需要 Prometheus 已在抓資料。

✅ 方法三：查看官方 repo 程式碼（適合深入研究）

VictoriaMetrics 是開源的，你可以在 GitHub 找到每個元件的源碼，裡面通常會直接使用 `**`prometheus/client_golang`**` 套件註冊 metrics。

GitHub 倉庫：https://github.com/VictoriaMetrics/VictoriaMetrics

你可以透過以下關鍵字搜尋 metrics：

- `**`prometheus.NewGauge`**`

- `prometheus.NewCounter`
- `prometheus.NewHistogram`
- `prometheus.MustRegister`

這些地方會定義 metrics 名稱、help 說明與 labels，非常完整，只是比較花時間。

✅ 方法四：社群文件與非官方整理

除了官方 GitHub 和文件，也有開源社群整理出一些 metrics 表（不一定最新），例如：

- Grafana Dashboard json 裡的 metrics 名稱
- 網友在部落格或 GitHub Gist 分享的 metrics 說明

例如這份非官方整理：

- Awesome VictoriaMetrics dashboards

❗為什麼「找不到所有 metrics 說明」？

VictoriaMetrics 的設計哲學偏向輕量與實作導向，所以不像 Prometheus 或 Kubernetes 那樣有一份很詳細的「所有 metrics 說明表」。

你可以：

- **自己用 script 定期打 `/metrics` 存成檔案比對**
- **搭配 `curl + grep` 來找特定指標名稱**

```
curl -s http://localhost:8429/metrics | grep vm_
```

✅ 額外建議：加上 `?format=prometheus` 或觀察 labels

部分 metrics 可以透過加上 query 參數格式化（如果支援），例如：

```
http://<host>:8429/metrics?format=prometheus
```

但通常預設就會是 Prometheus 格式。

如果你願意，我可以幫你：

- 幫你解析你目前 `/metrics` 中出現的所有指標名稱
- 幫你根據特定 component 整理出一份 metrics 說明表
- 幫你寫一段 script 抓指定 port 的 metrics 並分析

你目前是遇到找不到哪個 component 的 metrics 呢？我可以幫你查。

沒有metric:
// Copyright 2016 The Prometheus Authors
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
// http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in

```
writing, software
// distributed under the License is distributed on an "AS
IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY
KIND, either express or implied.
// See the License for the specific language governing
permissions and
// limitations under the License.

package remote

import (
    "bytes"
    "context"
    "errors"
    "fmt"
    "io"
    "net/http"
    "net/http/httptrace"
    "strconv"
    "strings"
    "time"

    "github.com/gogo/protobuf/proto"
    "github.com/golang/snappy"
    "github.com/prometheus/client_golang/prometheus"
    config_util "github.com/prometheus/common/config"
    "github.com/prometheus/common/model"
    "github.com/prometheus/common/version"
    "github.com/prometheus/sigv4"

"go.opentelemetry.io/contrib/instrumentation/net/http/h
ttptrace/otelhttptrace"

"go.opentelemetry.io/contrib/instrumentation/net/http/o
telhttp"
    "go.opentelemetry.io/otel"
    "go.opentelemetry.io/otel/trace"

    "github.com/prometheus/prometheus/config"
```

```
    "github.com/prometheus/prometheus/prompb"
    "github.com/prometheus/prometheus/storage"

"github.com/prometheus/prometheus/storage/remote/azuread"

"github.com/prometheus/prometheus/storage/remote/googleiam"
)

const (
    maxErrMsgLen = 1024

    RemoteWriteVersionHeader        = "X-Prometheus-Remote-Write-Version"
    RemoteWriteVersion1HeaderValue  = "0.1.0"
    RemoteWriteVersion20HeaderValue = "2.0.0"
    appProtoContentType             = "application/x-protobuf"
)

// Compression represents the encoding. Currently remote storage supports only
// one, but we experiment with more, thus leaving the compression scaffolding
// for now.
// NOTE(bwplotka): Keeping it public, as a non-stable help for importers to use.
type Compression string

const (
    // SnappyBlockCompression represents https://github.com/google/snappy/blob/2c94e11145f0b7b184b831577c93e5a41c4c0346/format_description.txt
    SnappyBlockCompression Compression = "snappy"
)

var (
    // UserAgent represents Prometheus version to use
```

```
for user agent header.
    UserAgent = version.PrometheusUserAgent()

    remoteWriteContentTypeHeaders =
map[config.RemoteWriteProtoMsg]string{
        config.RemoteWriteProtoMsgV1:
appProtoContentType, // Also application/x-
protobuf;proto=prometheus.WriteRequest but
simplified for compatibility with 1.x spec.
        config.RemoteWriteProtoMsgV2:
appProtoContentType +
";proto=io.prometheus.write.v2.Request",
    }

    AcceptedResponseTypes =
[]prompb.ReadRequest_ResponseType{
        prompb.ReadRequest_STREAMED_XOR_CHUNKS,
        prompb.ReadRequest_SAMPLES,
    }

    remoteReadQueriesTotal =
prometheus.NewCounterVec(
        prometheus.CounterOpts{
            Namespace: namespace,
            Subsystem: "remote_read_client",
            Name:      "queries_total",
            Help:      "The total number of remote read
queries.",
        },
        []string{remoteName, endpoint, "response_type",
"code"},
    )
    remoteReadQueries = prometheus.NewGaugeVec(
        prometheus.GaugeOpts{
            Namespace: namespace,
            Subsystem: "remote_read_client",
            Name:      "queries",
            Help:      "The number of in-flight remote read
queries.",
        },
```

```
        []string{remoteName, endpoint},
    )
    remoteReadQueryDuration =
prometheus.NewHistogramVec(
        prometheus.HistogramOpts{
            Namespace:                    namespace,
            Subsystem:                    "remote_read_client",
            Name:
"request_duration_seconds",
            Help:                         "Histogram of the
latency for remote read requests. Note that for
streamed responses this is only the duration of the
initial call and does not include the processing of the
stream.",
            Buckets:
append(prometheus.DefBuckets, 25, 60),
            NativeHistogramBucketFactor:     1.1,
            NativeHistogramMaxBucketNumber:  100,
            NativeHistogramMinResetDuration: 1 *
time.Hour,
        },
        []string{remoteName, endpoint,
"response_type"},
    )
)

func init() {
    prometheus.MustRegister(remoteReadQueriesTotal,
remoteReadQueries, remoteReadQueryDuration)
}

// Client allows reading and writing from/to a remote
HTTP endpoint.
type Client struct {
    remoteName string // Used to differentiate clients in
metrics.
    urlString  string // url.String()
    Client     *http.Client
    timeout    time.Duration
```

```
    retryOnRateLimit bool
    chunkedReadLimit uint64

    readQueries         prometheus.Gauge
    readQueriesTotal    *prometheus.CounterVec
    readQueriesDuration prometheus.ObserverVec

    writeProtoMsg    config.RemoteWriteProtoMsg
    writeCompression Compression // Not exposed by
ClientConfig for now.
}

// ClientConfig configures a client.
type ClientConfig struct {
    URL              *config_util.URL
    Timeout          model.Duration
    HTTPClientConfig config_util.HTTPClientConfig
    SigV4Config      *sigv4.SigV4Config
    AzureADConfig    *azuread.AzureADConfig
    GoogleIAMConfig  *googleiam.Config
    Headers          map[string]string
    RetryOnRateLimit bool
    WriteProtoMsg    config.RemoteWriteProtoMsg
    ChunkedReadLimit uint64
    RoundRobinDNS    bool
}

// ReadClient will request the STREAMED_XOR_CHUNKS
method of remote read but can
// also fall back to the SAMPLES method if necessary.
type ReadClient interface {
    Read(ctx context.Context, query *prompb.Query,
sortSeries bool) (storage.SeriesSet, error)
}

// NewReadClient creates a new client for remote read.
func NewReadClient(name string, conf *ClientConfig)
(ReadClient, error) {
    httpClient, err :=
config_util.NewClientFromConfig(conf.HTTPClientConfig,
```

```
"remote_storage_read_client")
    if err != nil {
        return nil, err
    }

    t := httpClient.Transport
    if len(conf.Headers) > 0 {
        t = newInjectHeadersRoundTripper(conf.Headers,
t)
    }
    httpClient.Transport = otelhttp.NewTransport(t)

    return &Client{
        remoteName:         name,
        urlString:          conf.URL.String(),
        Client:             httpClient,
        timeout:            time.Duration(conf.Timeout),
        chunkedReadLimit:    conf.ChunkedReadLimit,
        readQueries:
remoteReadQueries.WithLabelValues(name,
conf.URL.String()),
        readQueriesTotal:
remoteReadQueriesTotal.MustCurryWith(prometheus.La
bels{remoteName: name, endpoint:
conf.URL.String()}),
        readQueriesDuration:
remoteReadQueryDuration.MustCurryWith(prometheus.
Labels{remoteName: name, endpoint:
conf.URL.String()}),
    }, nil
}

// NewWriteClient creates a new client for remote write.
func NewWriteClient(name string, conf *ClientConfig)
(WriteClient, error) {
    var httpOpts []config_util.HTTPClientOption
    if conf.RoundRobinDNS {
        httpOpts =
[]config_util.HTTPClientOption{config_util.WithDialCont
extFunc(newDialContextWithRoundRobinDNS().dialCont
```

```
extFn())}
    }
    httpClient, err :=
config_util.NewClientFromConfig(conf.HTTPClientConfig,
"remote_storage_write_client", httpOpts...)
    if err != nil {
        return nil, err
    }
    t := httpClient.Transport

    if len(conf.Headers) > 0 {
        t = newInjectHeadersRoundTripper(conf.Headers,
t)
    }

    if conf.SigV4Config != nil {
        t, err =
sigv4.NewSigV4RoundTripper(conf.SigV4Config, t)
        if err != nil {
            return nil, err
        }
    }

    if conf.AzureADConfig != nil {
        t, err =
azuread.NewAzureADRoundTripper(conf.AzureADConfig,
t)
        if err != nil {
            return nil, err
        }
    }

    if conf.GoogleIAMConfig != nil {
        t, err =
googleiam.NewRoundTripper(conf.GoogleIAMConfig, t)
        if err != nil {
            return nil, err
        }
    }
```

```
    writeProtoMsg := config.RemoteWriteProtoMsgV1
    if conf.WriteProtoMsg != "" {
        writeProtoMsg = conf.WriteProtoMsg
    }
    httpClient.Transport = otelhttp.NewTransport(
        t,
        otelhttp.WithClientTrace(func(ctx context.Context)
*httptrace.ClientTrace {
            return otelhttptrace.NewClientTrace(ctx,
otelhttptrace.WithoutSubSpans())
        }))
    return &Client{
        remoteName:       name,
        urlString:        conf.URL.String(),
        Client:           httpClient,
        retryOnRateLimit: conf.RetryOnRateLimit,
        timeout:          time.Duration(conf.Timeout),
        writeProtoMsg:    writeProtoMsg,
        writeCompression: SnappyBlockCompression,
    }, nil
}

func newInjectHeadersRoundTripper(h
map[string]string, underlyingRT http.RoundTripper)
*injectHeadersRoundTripper {
    return &injectHeadersRoundTripper{headers: h,
RoundTripper: underlyingRT}
}

type injectHeadersRoundTripper struct {
    headers map[string]string
    http.RoundTripper
}

func (t *injectHeadersRoundTripper) RoundTrip(req
*http.Request) (*http.Response, error) {
    for key, value := range t.headers {
        req.Header.Set(key, value)
    }
    return t.RoundTripper.RoundTrip(req)
```

```
}

const defaultBackoff = 0

type RecoverableError struct {
    error
    retryAfter model.Duration
}

// Store sends a batch of samples to the HTTP endpoint,
the request is the proto marshalled
// and encoded bytes from codec.go.
func (c *Client) Store(ctx context.Context, req []byte,
attempt int) (WriteResponseStats, error) {
    httpReq, err := http.NewRequest(http.MethodPost,
c.urlString, bytes.NewReader(req))
    if err != nil {
        // Errors from NewRequest are from unparsable
URLs, so are not
        // recoverable.
        return WriteResponseStats{}, err
    }

    httpReq.Header.Add("Content-Encoding",
string(c.writeCompression))
    httpReq.Header.Set("Content-Type",
remoteWriteContentTypeHeaders[c.writeProtoMsg])
    httpReq.Header.Set("User-Agent", UserAgent)
    if c.writeProtoMsg ==
config.RemoteWriteProtoMsgV1 {
        // Compatibility mode for 1.0.
        httpReq.Header.Set(RemoteWriteVersionHeader,
RemoteWriteVersion1HeaderValue)
    } else {
        httpReq.Header.Set(RemoteWriteVersionHeader,
RemoteWriteVersion20HeaderValue)
    }

    if attempt > 0 {
        httpReq.Header.Set("Retry-Attempt",
```

```
strconv.Itoa(attempt))
    }

    ctx, cancel := context.WithTimeout(ctx, c.timeout)
    defer cancel()

    ctx, span := otel.Tracer("").Start(ctx, "Remote Store",
trace.WithSpanKind(trace.SpanKindClient))
    defer span.End()

    httpResp, err :=
c.Client.Do(httpReq.WithContext(ctx))
    if err != nil {
        // Errors from Client.Do are from (for example)
network errors, so are
        // recoverable.
        return WriteResponseStats{},
RecoverableError{err, defaultBackoff}
    }
    defer func() {
        _, _ = io.Copy(io.Discard, httpResp.Body)
        _ = httpResp.Body.Close()
    }()

    // TODO(bwplotka): Pass logger and emit debug on
error?
    // Parsing error means there were some response
header values we can't parse,
    // we can continue handling.
    rs, _ := ParseWriteResponseStats(httpResp)

    if httpResp.StatusCode/100 == 2 {
        return rs, nil
    }

    // Handling errors e.g. read potential error in the
body.
    // TODO(bwplotka): Pass logger and emit debug on
error?
    body, _ := io.ReadAll(io.LimitReader(httpResp.Body,
```

```
maxErrMsgLen))
    err = fmt.Errorf("server returned HTTP status %s:
%s", httpResp.Status, body)

    if httpResp.StatusCode/100 == 5 ||
        (c.retryOnRateLimit && httpResp.StatusCode ==
http.StatusTooManyRequests) {
        return rs, RecoverableError{err,
retryAfterDuration(httpResp.Header.Get("Retry-
After"))}
    }
    return rs, err
}

// retryAfterDuration returns the duration for the Retry-
After header. In case of any errors, it
// returns the defaultBackoff as if the header was never
supplied.
func retryAfterDuration(t string) model.Duration {
    parsedDuration, err := time.Parse(http.TimeFormat,
t)
    if err == nil {
        s := time.Until(parsedDuration).Seconds()
        return model.Duration(s) *
model.Duration(time.Second)
    }
    // The duration can be in seconds.
    d, err := strconv.Atoi(t)
    if err != nil {
        return defaultBackoff
    }
    return model.Duration(d) *
model.Duration(time.Second)
}

// Name uniquely identifies the client.
func (c *Client) Name() string {
    return c.remoteName
}
```

```
// Endpoint is the remote read or write endpoint.
func (c *Client) Endpoint() string {
    return c.urlString
}

// Read reads from a remote endpoint. The sortSeries
parameter is only respected in the case of a sampled
response;
// chunked responses arrive already sorted by the
server.
func (c *Client) Read(ctx context.Context, query
*prompb.Query, sortSeries bool) (storage.SeriesSet,
error) {
    c.readQueries.Inc()
    defer c.readQueries.Dec()

    req := &prompb.ReadRequest{
        // TODO: Support batching multiple queries into
one read request,
        // as the protobuf interface allows for it.
        Queries:              []*prompb.Query{query},
        AcceptedResponseTypes: AcceptedResponseTypes,
    }
    data, err := proto.Marshal(req)
    if err != nil {
        return nil, fmt.Errorf("unable to marshal read
request: %w", err)
    }

    compressed := snappy.Encode(nil, data)
    httpReq, err := http.NewRequest(http.MethodPost,
c.urlString, bytes.NewReader(compressed))
    if err != nil {
        return nil, fmt.Errorf("unable to create request:
%w", err)
    }
    httpReq.Header.Add("Content-Encoding", "snappy")
    httpReq.Header.Add("Accept-Encoding", "snappy")
    httpReq.Header.Set("Content-Type", "application/x-
protobuf")
```

```
    httpReq.Header.Set("User-Agent", UserAgent)
    httpReq.Header.Set("X-Prometheus-Remote-Read-
Version", "0.1.0")

    ctx, cancel := context.WithTimeout(ctx, c.timeout)

    ctx, span := otel.Tracer("").Start(ctx, "Remote Read",
trace.WithSpanKind(trace.SpanKindClient))
    defer span.End()

    start := time.Now()
    httpResp, err :=
c.Client.Do(httpReq.WithContext(ctx))
    if err != nil {
        cancel()
        return nil, fmt.Errorf("error sending request: %w",
err)
    }

    if httpResp.StatusCode/100 != 2 {
        // Make an attempt at getting an error message.
        body, _ := io.ReadAll(httpResp.Body)
        _ = httpResp.Body.Close()

        cancel()
        return nil, fmt.Errorf("remote server %s returned
http status %s: %s", c.urlString, httpResp.Status,
string(body))
    }

    contentType := httpResp.Header.Get("Content-Type")

    switch {
    case strings.HasPrefix(contentType, "application/x-
protobuf"):

c.readQueriesDuration.WithLabelValues("sampled").Ob
serve(time.Since(start).Seconds())
        c.readQueriesTotal.WithLabelValues("sampled",
strconv.Itoa(httpResp.StatusCode)).Inc()
```

```
        ss, err := c.handleSampledResponse(req,
httpResp, sortSeries)
        cancel()
        return ss, err
    case strings.HasPrefix(contentType, "application/x-
streamed-protobuf;
proto=prometheus.ChunkedReadResponse"):

c.readQueriesDuration.WithLabelValues("chunked").Ob
serve(time.Since(start).Seconds())

        s := NewChunkedReader(httpResp.Body,
c.chunkedReadLimit, nil)
        return NewChunkedSeriesSet(s, httpResp.Body,
query.StartTimestampMs, query.EndTimestampMs,
func(err error) {
            code := strconv.Itoa(httpResp.StatusCode)
            if !errors.Is(err, io.EOF) {
                code = "aborted_stream"
            }
            c.readQueriesTotal.WithLabelValues("chunked",
code).Inc()
            cancel()
        }), nil
    default:

c.readQueriesDuration.WithLabelValues("unsupported")
.Observe(time.Since(start).Seconds())

c.readQueriesTotal.WithLabelValues("unsupported",
strconv.Itoa(httpResp.StatusCode)).Inc()
        cancel()
        return nil, fmt.Errorf("unsupported content type:
%s", contentType)
    }
}

func (c *Client) handleSampledResponse(req
*prompb.ReadRequest, httpResp *http.Response,
sortSeries bool) (storage.SeriesSet, error) {
```

```
    compressed, err := io.ReadAll(httpResp.Body)
    if err != nil {
        return nil, fmt.Errorf("error reading response. HTTP
status code: %s: %w", httpResp.Status, err)
    }
    defer func() {
        _, _ = io.Copy(io.Discard, httpResp.Body)
        _ = httpResp.Body.Close()
    }()

    uncompressed, err := snappy.Decode(nil,
compressed)
    if err != nil {
        return nil, fmt.Errorf("error reading response: %w",
err)
    }

    var resp prompb.ReadResponse
    err = proto.Unmarshal(uncompressed, &resp)
    if err != nil {
        return nil, fmt.Errorf("unable to unmarshal
response body: %w", err)
    }

    if len(resp.Results) != len(req.Queries) {
        return nil, fmt.Errorf("responses: want %d, got
%d", len(req.Queries), len(resp.Results))
    }

    // This client does not batch queries so there's
always only 1 result.
    res := resp.Results[0]

    return FromQueryResult(sortSeries, res), nil
}
```

你貼的是 Prometheus 的 remote storage client 的 Go 原始碼，這段程式碼負責與 Remote Write/Read target（像是 VictoriaMetrics）溝通，但不是 VictoriaMetrics 本身的實作，因此 **你不會在這份程式碼中找到 VictoriaMetrics 的 metrics 定義**。

如果你遇到的問題是：「部署了 VictoriaMetrics 但 `/metrics` 沒有資料或 metric 數量很少」，我們可以分幾個層面來排查與解釋。

🔍 問題釐清：你是遇到哪一種情況？

1. **無法打開 `/metrics` 頁面？**
    - 是連不到？還是 404？
2. **`/metrics` 有內容但沒看到你期待的 metrics？**
    - 只看到 Go 的 runtime metrics（例如 `go_gc_duration_seconds`）？
3. **你貼的是 Prometheus 原始碼，是不是你在做遠端讀寫時看不到對應 metrics？**
    - 想追蹤某個 remote write client 的指標？例如 Prometheus 對 VM 的寫入延遲？

請確認是哪一種情況，我可以更精準地協助你。下面先假設你是想查「VictoriaMetrics 各 component 的內建 metrics」。

🔧 正確取得 VictoriaMetrics metrics 的方式

VictoriaMetrics 的各個 component（如 `vminsert`, `vmstorage`, `vmselect`, `vmagent`）會在其 HTTP 伺服器的 `/metrics` endpoint 暴露其內部狀態，用來給 Prometheus 抓。

Component	預設 Port	指標位置
`vminsert`	8480	`/metrics`
`vmselect`	8481	`/metrics`
`vmstorage`	8482	`/metrics`
`vmagent`	8429	`/metrics`
`victoria-metrics single-node`	8428	`/metrics`

✅ 正確測試方式：

```
curl http://<host>:<port>/metrics
```

例如：

```
curl http://localhost:8428/metrics | grep vm_
```

🧪 常見問題排查

1️⃣ `/metrics` 是空的？

- 確認你的 component 正常啟動（`ps aux | grep vmagent` 等）
- 確認不是被防火牆或 ingress 擋掉
- 你可能用了 reverse proxy 忘了 expose `/metrics`

2️⃣ `/metrics` 只有 Go 的 runtime metrics？

- 這代表你的 component 是空轉狀態
- 例如 `vminsert` 沒有實際接收到任何 Remote Write 的資料 ⇒ 就沒產生 ingestion metrics
- 例如 `vmagent` 沒有 job 被配置 ⇒ 就沒有 scrape 的 metrics

✅ 建議你測試：

```
curl http://localhost:8429/metrics | grep vmagent_
```

或是用：

```
grep -E '^(vm|prometheus|go|process)_' /metrics
```

🔍 如果你是想「找出所有能出現的 metrics 名稱」怎麼辦？

VictoriaMetrics 官方其實沒整理出完整 metrics 文件，但你可以用以下幾種方式「間接收集」所有可能的 metrics：

✅ 方法一：看 source code（真的要找只能這樣）

在 VictoriaMetrics GitHub 的每個 component 內部，都會註冊 metrics。

例如：

```
metrics.GetOrCreateCounter("vm_rows_inserted_total")
metrics.GetOrCreateHistogram("vm_http_request_duration_seconds")
```

用 GitHub 全域搜尋 `GetOrCreate` 可以抓到所有 metrics。

✅ 方法二：跑壓測（benchmark）+ `/metrics` 抓 snapshot

- 跑寫入測試、查詢測試、各種配置
- 用 `curl` 抓 `/metrics` 回來存檔
- 跟 baseline 比對就可以知道哪些 metrics 有增加出現

📦 補充資源

- VictoriaMetrics 官方 metrics FAQ：
  - https://docs.victoriametrics.com/#frequently-asked-questions
- GitHub 原始碼 metrics 搜尋建議：
  - https://github.com/VictoriaMetrics/VictoriaMetrics/search?q=GetOrCreate

---

🎯 總結：VictoriaMetrics metrics 查不到的原因

原因	解法
`/metrics` 頁面沒開起來	確認 port、防火牆、是否有啟動
metrics 很少或只有 Go 相關	沒有實際工作量（沒收資料、沒查詢）
想找所有可能 metrics 名稱	從 GitHub 搜尋 `metrics.GetOrCreate*`
你在看 Prometheus 的 remote client code	那不是 VictoriaMetrics 的部分，看不到對方 expose 的 metrics

---

如果你能補充一下目前遇到哪個 component、部署架構、具體「哪個 metrics 查不到」，我可以幫你對症下藥找出來！也可以幫你寫 script 一次性抓出所有可能的 `/metrics` name。你方便提供目前是哪一個 component 或部署架構嗎？

我網頁上有收到metric，但沒有查到metric代表的意義

感謝補充！你這樣一說我懂了：你 **打開 `/metrics` 網頁是 OK 的，也看到有一堆指標（如 `vm_xxx_xxx_total`）**，但問題是：

> *“💬「我不知道這些 metric 是什麼意思、代表什麼行為或事件。」”*

這其實是很多人在使用 **VictoriaMetrics** 時會遇到的共同問題，因為官方文件對 metrics 解釋有限（不像 Prometheus 有完整 reference），但我會教你幾種方法來搞懂這些 metric 的「意思」，並附上一些常見的指標解釋。

✅ 方法一：從 metric 名稱推測意義（搭配行為觀察）

VictoriaMetrics 的 metrics 都是以 `vm_` 開頭，有些結尾是 `_total`、`_bytes`，`_duration_seconds`，`_requests`，`_errors_total`，這些慣例可對照 Prometheus 的命名規範推測：

尾碼	類型	說明
`_total`	Counter	數量總和，會持續遞增
`_duration_seconds`	Histogram/Summary	某類操作的耗時
`_bytes`	Counter	傳輸或儲存的位元組數

尾碼	類型	說明
`_requests`	Counter	發出的請求次數
`_errors_total`	Counter	發生錯誤的次數

📌 例子：`vm_rows_inserted_total`

- 📋 類型：`Counter`
- ⏱️ 意義：VictoriaMetrics 收到多少 time series data row 被成功寫入（通常來自 remote write）
- 🔍 使用時機：看你的寫入量

✅ 方法二：從 Grafana Dashboard 借助現成解釋

VictoriaMetrics 社群提供了很多 Grafana dashboard，可以直接參考它們用哪些 metrics、搭配什麼解釋文字（panel title、description），例如：

👉 VictoriaMetrics Grafana Dashboards 官方整理

建議下載或套用這幾個 dashboard：

- `VictoriaMetrics Single Node Overview`
- `Cluster Overview`
- `vmagent Overview`

裡面會寫出像：

Metric 名稱	解釋
`vm_ingest_ingested_samples_total`	Number of ingested samples（寫入總樣本數）
`vm_cache_entries`	Entries in internal cache
`vm_http_requests_total`	HTTP 請求次數

✅ 方法三：查 GitHub 原始碼中的 metric 註解

VictoriaMetrics 的 metrics 都是透過：

```
metrics.GetOrCreateCounter("vm_rows_inserted_total")
```

來註冊。雖然大多沒有寫註解，但有些地方會寫在同一段 code 的 comment 裡，搜尋建議：

- 進 GitHub 倉庫：https://github.com/VictoriaMetrics/VictoriaMetrics
- 用全倉庫搜尋關鍵字：
  - `GetOrCreateCounter("vm_`
  - `metrics.NewHistogram`
  - `metrics.GetOrCreateSummary`

✅ 方法四：看你是什麼 component（vmagent / vminsert / vmselect / vmstorage）

每個 component metrics 意義不一樣。以下是一些常見 component 的 metric 說明：

`vmagent`

Metric	說明
`vm_promscrape_scrapes_total`	抓取成功的次數
`vm_promscrape_scraped_samples_total`	抓到的 sample 數
`vm_promscrape_target_interval_length_seconds`	每個 target 的抓取間隔
`vm_promscrape_target_last_scrape_duration_seconds`	抓取花了多少時間
`vm_promscrape_scrapes_failed_total`	抓取失敗次數（需監控）

`vminsert`

Metric	說明
`vm_rows_inserted_total`	實際寫入成功的 row 數
`vm_rows_per_insert`	每次 insert row 數
`vm_http_requests_total{path="/insert"}`	接收到的 insert 請求數

`vmselect`

Metric	說明
`vm_search_queries_total`	查詢次數
`vm_search_duration_seconds`	查詢延遲
`vm_http_requests_total{path="/select"}`	查詢 API 次數

✅ 想知道某個特定 metric 是什麼意思？你可以這樣問我：

例如你看到：

```
vm_http_request_duration_seconds_bucket{path="/insert", le="0.5"} 1234
```

你可以問我：

> *“這個 `vm_http_request_duration_seconds_bucket{path="/insert"}` 是什麼意思？”*

我可以幫你解析它的類型（Histogram）、用途（HTTP insert request 的延遲分佈）、意義（第 0.5 秒以內請求的次數）等等。

🎁 Bonus：我可以幫你生成指標解釋表

如果你願意提供你 `/metrics` 內容，我可以幫你整理成如下表格：

Metric 名稱	類型	解釋	建議觀測方式
vm_rows_inserted_total	Counter	總寫入 row 數	應遞增，可搭配 rate() 看寫入速率
vm_http_requests_total{path="/insert"}	Counter	接收寫入請求數	看流量變化、錯誤率

📌 結論整理

- `/metrics` 可以看到指標，但官方沒有詳細說明

- 可用名稱推測、Grafana dashboard、GitHub 搜尋、或問我來搞懂每個 metrics
- 若你貼出你看到的某段 metrics，我可以幫你完整解釋每一行的用途與使用建議！

👉 你方便貼幾行你 `/metrics` 中不懂的 metrics 嗎？我幫你一行一行解釋！你也可以告訴我是用哪個 component，我會根據那個環境幫你拆解含意。