



Prometheus vs VictoriaMetrics



詳細說明怎麼比較promethe、victoriameetric-cluster



Prometheus 和 VictoriaMetrics Cluster 都是用來監控和存儲時序數據 (TSDB, Time-Series Database) 的解決方案，兩者在架構、效能、擴展性、功能支援上有所不同。以下是詳細的比較方式：

1. 架構對比

| 特色 | Prometheus | VictoriaMetrics Cluster |
|------|---|------------------------------|
| 架構 | 單節點存儲，可透過聯邦集群 (Federation) 或 Thanos / Cortex 進行擴展 | 分布式集群架構，具備原生水平擴展能力 |
| 存儲引擎 | 基於 TSDB (Chunk-based Storage) | 自家設計的存儲引擎，使用 LSM-Tree |
| 擴展性 | 需借助 Thanos / Cortex 來支持高可用和水平擴展 | 原生支持多節點水平擴展 |
| 高可用性 | 透過多個 Prometheus 節點+Thanos Query | 內建 HA (High Availability) 機制 |

2. 效能比較

| 特色 | Prometheus | VictoriaMetrics Cluster |
|-------|-----------------------|--------------------------------|
| 寫入吞吐量 | 單節點處理數百萬時間序列，但受限於本機存儲 | 可支持更高的寫入吞吐量，透過 sharding 進行負載均衡 |
| 查詢性能 | 受限於單節點，隨著數據增長查詢時間會增加 | 基於分布式查詢，能更快處理大量數據 |





| 特色 | Prometheus | VictoriaMetrics Cluster |
|-----|-----------------------|----------------------------|
| 壓縮率 | 約 1-2 byte per sample | 0.3 byte per sample, 壓縮率更高 |

VictoriaMetrics 通常比 Prometheus 更省資源，因為它有更優化的壓縮演算法，並支援更大規模的時序數據存儲。

3. 部署與維運

| 特色 | Prometheus | VictoriaMetrics Cluster |
|------|---|--|
| 安裝 | 簡單，單一二進制文件運行 | 較複雜，需要配置 multiple components (vmstorage, vminsert, vmselect) |
| 難度 | | |
| 擴展方式 | 透過聯邦集群 (Federation) 或 Thanos / Cortex 來擴展 | 可直接通過增加節點來擴展 |
| 維護成本 | 單節點架構易維護，但需要定期刪除過期數據 | 分布式系統需要專業維運，但存儲管理較佳 |

Prometheus 適合小型部署，而 VictoriaMetrics Cluster 更適合大規模分布式環境。

4. 功能支援

| 特色 | Prometheus | VictoriaMetrics Cluster |
|--------------|--|--|
| 原生 PromQL 支援 | <input checked="" type="checkbox"/> 完整支援 | <input checked="" type="checkbox"/> 完整支援，並提供額外擴展功能 |
| 歷史數據導入 | 需透過 remote_write 或 TSDB API | 可使用 CSV、JSON、OpenMetrics 等格式直接導入 |
| 遠端存儲 | 透過 remote_write / remote_read | 內建高效遠端存儲支持 |
| 內存占用 | 較高，需手動優化配置 | 相對較低，存儲壓縮更佳 |





VictoriaMetrics Cluster 在數據存儲和遠端存儲方面較有優勢，而 Prometheus 則擁有更完整的生態系統。

5. 適用場景

- **Prometheus**

- 適合 **小型/中型監控環境**
- 適合 **簡單部署**
- 需要 **與 Kubernetes 結合** (如 Prometheus Operator)

- **VictoriaMetrics Cluster**

- 適合 **大規模時序數據存儲**
 - 適合 **高吞吐量場景**
 - 適合 **需要高可用 HA 架構**
-

6. 總結

如果你的需求是：

- 監控少量應用、單機架構 → **Prometheus**
- 需要擴展性好、查詢更快、節省存儲空間 → **VictoriaMetrics Cluster**

如果已有 Prometheus，可以考慮 **Prometheus + VictoriaMetrics**

(**remote_write**)，利用 VictoriaMetrics Cluster 作為長期存儲解決方案。

這樣的比較方式應該能幫助你決定適合的監控系統！🚀



以繁體中文詳細說明:我用docker-compose 建立了vmstorage, vmlinser, vmselect, vmagent, vmalert 各一個node，請問怎麼知道使用了多少cpu、mem





要監控您使用 Docker Compose 部署的 VictoriaMetrics 組件（如 vmstorage、vminsert、vmselect、vmagent、vmalert）各自的 CPU 和記憶體使用情況，您可以採取以下方法：

1. 使用 `docker stats` 指令

Docker 提供了 `docker stats` 指令，可實時顯示所有運行中容器的資源使用情況，包括 CPU、記憶體、網路 I/O 等。

查看所有運行中容器的資源使用情況：

```
docker stats
```

此指令會持續刷新，顯示每個容器的以下資訊：

- **CONTAINER ID**：容器 ID
- **NAME**：容器名稱
- **CPU %**：CPU 使用率
- **MEM USAGE / LIMIT**：記憶體使用量和限制
- **MEM %**：記憶體使用率
- **NET I/O**：網路輸入/輸出
- **BLOCK I/O**：磁碟輸入/輸出
- **PIDS**：進程數

如果只想查看特定容器的資源使用情況，可以在指令後指定容器的名稱或 ID：

```
docker stats <container_name_or_id>
```

例如，查看名為 `vmstorage` 的容器：

```
docker stats vmstorage
```





要僅查看當前的資源使用狀態而不持續刷新，使用 `--no-stream` 選項：

```
docker stats --no-stream
```

這將顯示一次性結果，然後退出。

2. 使用 `docker inspect` 指令

`docker inspect` 指令可提供容器的詳細資訊，包括資源配置和使用情況。

查看特定容器的詳細資訊：

```
docker inspect <container_name_or_id>
```

此指令會輸出容器的 JSON 格式詳細資訊。您可以使用 `grep` 或其他工具過濾與記憶體相關的資訊：

```
docker inspect <container_name_or_id> | grep -i Memory
```

這將顯示與記憶體相關的配置和使用資訊。

3. 使用 `/sys/fs/cgroup` 檔案系統

Docker 容器的資源使用資訊也可以通過宿主機的 cgroup 檔案系統查看。

查看特定容器的記憶體使用情況：

1. 首先，找到容器的完整 ID：

```
docker ps --no-trunc
```





2. 然後，查看該容器的記憶體使用：

```
cat /sys/fs/cgroup/memory/docker/<container_id>/memory.usage_in_bytes
```

這將以位元組為單位顯示容器的當前記憶體使用量。

查看容器的 CPU 使用情況：

```
cat /sys/fs/cgroup/cpu/docker/<container_id>/cpuacct.usage
```

這將顯示容器的累積 CPU 使用時間（以奈秒為單位）。

4. 使用第三方監控工具

為了更直觀和全面地監控容器的資源使用情況，您可以考慮使用以下第三方工具：

- **cAdvisor**：專門用於監控容器資源使用，包括 CPU、記憶體、磁碟和網路。
- **Prometheus + Grafana**：結合使用，提供強大的監控和可視化功能。您可以配置 Prometheus 來抓取 Docker 容器的指標，並使用 Grafana 進行可視化。
- **Portainer**：提供 Docker 容器管理和資源使用的圖形介面。

這些工具可以提供更詳細的歷史數據和圖表，方便您分析和監控容器的性能。

通過上述方法，您可以有效地監控 Docker 容器的 CPU 和記憶體使用情況，確保您的 VictoriaMetrics 組件運行在最佳狀態。

資料來源





使用 Prometheus + Grafana 監控 Docker 容器 (VictoriaMetrics 組件)

Prometheus 和 Grafana 是最常用的監控和可視化工具，特別適用於監控 Docker 容器的 CPU、記憶體、網絡流量等資源使用狀況。以下是詳細的安裝與設定步驟：

1. 安裝與設定 Prometheus

Prometheus 是一個時序數據庫 (TSDB)，可以從 Docker 容器獲取監控數據。

1.1 建立 `docker-compose.yml`

在您的專案目錄下，建立一個 `docker-compose.yml` 檔案，並加入以下內容：

```
version: '3.7'
services:
  prometheus:
    image: prom/prometheus
    container_name: prometheus
    restart: always
    ports:
      - "9090:9090"
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    networks:
      - monitoring

  node-exporter:
    image: prom/node-exporter
    container_name: node-exporter
    restart: always
    ports:
      - "9100:9100"
    networks:
```





```
- monitoring

cadvisor:
  image: gcr.io/cadvisor/cadvisor
  container_name: cadvisor
  restart: always
  privileged: true
  ports:
    - "8080:8080"
  volumes:
    - "/:/rootfs:ro"
    - "/var/run:/var/run:rw"
    - "/sys:/sys:ro"
    - "/var/lib/docker/:/var/lib/docker:ro"
  networks:
    - monitoring

grafana:
  image: grafana/grafana
  container_name: grafana
  restart: always
  ports:
    - "3000:3000"
  networks:
    - monitoring
  volumes:
    - grafana-data:/var/lib/grafana
  environment:
    - GF_SECURITY_ADMIN_PASSWORD=admin

networks:
  monitoring:

volumes:
  grafana-data:
```

2. 設定 Prometheus

在 `docker-compose.yml` 所在的目錄下，建立 `prometheus.yml` 配置文件，內容如下：



```
global:  
  scrape_interval: 5s  
  
scrape_configs:  
  - job_name: 'prometheus'  
    static_configs:  
      - targets: ['localhost:9090']  
  
  - job_name: 'node-exporter'  
    static_configs:  
      - targets: ['node-exporter:9100']  
  
  - job_name: 'cadvisor'  
    static_configs:  
      - targets: ['cadvisor:8080']
```

3. 啟動監控系統

在終端機執行以下指令：

```
docker-compose up -d
```

- ` -d` 參數表示以背景模式運行。
- 運行後，可透過 `http://localhost:9090` 存取 Prometheus。

4. 設定 Grafana

4.1 進入 Grafana

在瀏覽器打開 `http://localhost:3000`，預設帳號密碼如下：

- 帳號：`admin`
- 密碼：`admin`（第一次登入會要求更改密碼）





4.2 新增 Prometheus 資料來源

1. 點擊左側「齒輪」圖標 (⚙) → `Data Sources`。
2. 點擊「Add data source」。
3. 選擇 `Prometheus`。
4. 在 `URL` 欄位填入：

```
http://prometheus:9090
```

5. 點擊「Save & Test」確認連接成功。

5. 建立監控儀表板

5.1 匯入現成的 Dashboard

1. 點擊左側「四方形圖標」 → `Dashboards`。
2. 點擊「Import」。
3. 在「Import via Grafana.com」輸入以下 ID：
 - **Docker 監控儀表板 ID** : `893` (Docker & system metrics)
 - **Node Exporter 監控 ID** : `1860` (Linux 監控)
4. 點擊「Load」，選擇 Prometheus 作為數據來源，然後點擊「Import」。

5.2 建立自訂 Dashboard

1. 點擊「+ Create」 → `Dashboard`。
2. 點擊「Add new panel」。
3. 選擇 `Prometheus` 作為數據來源。
4. 在 `Query` 欄位輸入：

```
container_memory_usage_bytes{container_label_com_docker_compose_service="vmstorage"}
```

這條查詢會顯示 `vmstorage` 容器的記憶體使用量。

5. 點擊「Save」，命名儀表板。





6. 監控 VictoriaMetrics

VictoriaMetrics 組件（vmstorage、vminsert、vmselect、vmagent、vmalert）本身也能暴露 Prometheus 指標，可在 `prometheus.yml` 中新增：

```
- job_name: 'victoriameetrics'  
  static_configs:  
    - targets: ['vmstorage:8428', 'vminsert:8480', 'vmselect:8481',  
      'vmagent:8429', 'vmalert:8880']
```

然後重啟 Prometheus：

```
docker-compose restart prometheus
```

7. 查看監控結果

- Prometheus 查詢界面：`<http://localhost:9090>`
- Grafana 儀表板：`<http://localhost:3000>`
- cAdvisor (容器監控)：`<http://localhost:8080>`

8. 查詢 VictoriaMetrics 監控數據

在 Prometheus UI 中，可使用以下 PromQL 查詢 VictoriaMetrics 相關的指標：

8.1 監控 CPU 使用率

```
rate(process_cpu_seconds_total{job="victoriameetrics"}[5m])
```





8.2 監控記憶體使用

```
container_memory_usage_bytes{job="victoriametrics"}
```

8.3 監控磁碟 IO

```
rate(container_fs_io_time_seconds_total{job="victoriametrics"}[5m])
```

8.4 監控網路流量

```
rate(container_network_receive_bytes_total{job="victoriametrics"}[5m])
```

9. 總結

1. 安裝 **Prometheus**、**Grafana**、**cAdvisor**、**Node Exporter**
2. 配置 `prometheus.yml` 來收集 Docker 和 VictoriaMetrics 的監控數據
3. 啟動 `docker-compose up -d`
4. 進入 Grafana，設定 Prometheus 作為數據來源
5. 匯入或建立監控 Dashboard
6. 查詢 CPU、記憶體、磁碟和網絡使用情況
7. 使用 PromQL 查詢 VictoriaMetrics 組件的指標

這樣就能成功透過 **Prometheus + Grafana** 監控 **VictoriaMetrics** (**vmstorage**、**vminsert**、**vmselect**、**vmagent**、**vmalert**) 及其他 Docker 容器的資源使用情況！



以繁體中文詳細說明:還有哪些kpi值得與prometheus比較



Powered by [ChatGPT Exporter](#)



使用 Prometheus 監控系統時值得比較的 KPI（關鍵績效指標）

除了基本的 CPU、記憶體、磁碟和網絡使用率外，還有許多其他的關鍵指標（KPI）值得監測，特別是在使用 **Prometheus** 來監控 **VictoriaMetrics** 或其他系統時。這些 KPI 可以幫助分析系統效能、容量規劃，以及發現潛在的問題。

1. 資源使用相關 KPI

這些 KPI 主要衡量系統資源的使用情況，確保應用不會超出硬體資源限制。

| KPI | Prometheus 查詢語法 | 說明 |
|----------|--|-------------------|
| CPU 使用率 | `rate(process_cpu_seconds_total[5m])` | 計算 5 分鐘內的 CPU 使用率 |
| 記憶體使用量 | `process_resident_memory_bytes` | 監控應用的實際記憶體使用量 |
| 記憶體使用百分比 | `(process_resident_memory_bytes / process_virtual_memory_bytes) * 100` | 監控記憶體使用比例 |
| 磁碟 I/O | `rate(node_disk_io_time_seconds_total[5m])` | 監測磁碟讀寫操作時間 |
| 磁碟空間剩餘量 | `node_filesystem_free_bytes` | 監控剩餘磁碟空間 |
| 磁碟使用率 | `(node_filesystem_size_bytes - node_filesystem_free_bytes) / node_filesystem_size_bytes * 100` | 計算磁碟使用百分比 |
| 網絡流量 | `rate(node_network_receive_bytes_total[5m])` | 監控網絡接收速率 |
| 網絡吞吐量 | `rate(node_network_transmit_bytes_total[5m])` | 監控網絡發送速率 |

這些指標適用於監測 **VictoriaMetrics** 伺服器（`vmstorage`，`vminsert`，`vmselect`）或其他應用伺服器的資源狀況。





2. 容器與 Kubernetes 相關 KPI

如果使用 **Docker** 或 **Kubernetes**，這些指標可以幫助監控容器的狀況。

| KPI | Prometheus 查詢語法 | 說明 |
|------------|--|--------------------|
| 容器 CPU 使用率 | `rate(container_cpu_usage_seconds_total[5m])` | 監控 5 分鐘內容器的 CPU 使用 |
| 容器記憶體使用量 | `container_memory_usage_bytes` | 監測容器當前使用的記憶體 |
| 容器磁碟讀取量 | `rate(container_fs_reads_bytes_total[5m])` | 監控容器的磁碟讀取流量 |
| 容器磁碟寫入量 | `rate(container_fs_writes_bytes_total[5m])` | 監控容器的磁碟寫入流量 |
| 容器重啟次數 | `increase(kube_pod_container_status_restarts_total[5m])` | 監控 5 分鐘內容器的重啟次數 |

這些指標適用於監控 **Docker** 或 **Kubernetes** 環境下的 VictoriaMetrics 組件，如 `vmstorage`, `vminsert`, `vmselect`, `vmagent`, `vmalert`。

3. VictoriaMetrics 相關 KPI

VictoriaMetrics 提供許多內部指標，可以直接監控時序數據庫的效能。

| KPI | Prometheus 查詢語法 | 說明 |
|---------------------------|------------------------------------|----------------------------------|
| 存儲數據量 (Time Series 數量) | `vm_tsdb_time_series` | 監測當前存儲的時序數據量 |
| 數據寫入速率 | `rate(vm_inserted_rows_total[5m])` | 監測 5 分鐘內寫入 VictoriaMetrics 的數據行數 |





| KPI | Prometheus 查詢語法 | 說明 |
|-----------|--|-------------------|
| 數據查詢速率 | <code>rate(vm_cache_requests_total[5m])`</code> | 監測 5 分鐘內的查詢請求數量 |
| 查詢延遲 | <code>`rate(vm_cache_misses_total[5m]) / rate(vm_cache_requests_total[5m])`</code> | 計算查詢緩存命中率 |
| HTTP 請求數量 | <code>rate(vm_http_requests_total[5m])`</code> | 監控 HTTP API 請求的總數 |
| HTTP 錯誤率 | <code>rate(vm_http_requests_total{status=~"5.."}[5m]) / rate(vm_http_requests_total[5m])`</code> | 計算 HTTP 5XX 錯誤的比率 |

這些指標適用於監控 **VictoriaMetrics** 的效能表現，確保數據寫入和查詢的穩定性。

4. 應用層 KPI

這些 KPI 適用於監控 **應用程式**，例如微服務、API 伺服器等。

| KPI | Prometheus 查詢語法 | 說明 |
|------------|---|--------------------|
| HTTP 請求數量 | <code>rate(http_requests_total[5m])`</code> | 監測 5 分鐘內 HTTP 請求數量 |
| HTTP 錯誤率 | <code>rate(http_requests_total{status=~"5.."}[5m]) / rate(http_requests_total[5m])`</code> | 計算 HTTP 5XX 錯誤的比率 |
| 請求延遲 (P99) | <code>histogram_quantile(0.99, rate(http_request_duration_seconds_bucket[5m]))`</code> | 監控 P99 延遲時間 |
| 請求延遲 (平均值) | <code>rate(http_request_duration_seconds_sum[5m]) / rate(http_request_duration_seconds_count[5m])`</code> | 計算 HTTP 請求的平均延遲 |

這些指標適用於監控 **API 伺服器** 或 **微服務** 的效能。





5. 監控告警 KPI

這些 KPI 用於設定 **告警條件**，當指標超過閾值時，觸發告警。

| KPI | Prometheus 查詢語法 | 告警條件 |
|------------|---|-------------------|
| 高 CPU 使用率 | `rate(process_cpu_seconds_total[5m]) > 0.8` | CPU 使用率超過 80% |
| 高記憶體使用率 | `process_resident_memory_bytes / process_virtual_memory_bytes > 0.9` | 記憶體使用率超過 90% |
| 磁碟剩餘空間過低 | `node_filesystem_free_bytes < 10 * 1024 * 1024 * 1024` | 剩餘磁碟空間少於 10GB |
| HTTP 錯誤率過高 | `rate(http_requests_total{status=~"5.."}[5m]) / rate(http_requests_total[5m]) > 0.05` | HTTP 5XX 錯誤率超過 5% |
| 查詢延遲過高 | `histogram_quantile(0.99, rate(http_request_duration_seconds_bucket[5m])) > 1` | P99 延遲超過 1 秒 |

這些 KPI 可搭配 **vmalert** 或 **Alertmanager** 來設定告警通知。

6. 總結

值得監控的 KPI 類別

1. **資源使用** (CPU、記憶體、磁碟、網絡)
2. **容器/Kubernetes** (容器資源、重啟次數)
3. **VictoriaMetrics 指標** (數據量、查詢效能)
4. **應用層指標** (HTTP 延遲、錯誤率)
5. **告警 KPI** (超過閾值時觸發警報)

這些 KPI 可搭配 **Prometheus** 和 **Grafana** 來監控 **VictoriaMetrics** 及整個系統的運行狀況，確保服務穩定運行並及時發現問題！🚀





以繁體中文詳細說明:我用docker-compose 建立了vmstorage, vminsert, vmselect, vmagent, vmalert 各一個node， 請問怎麼跟使用單個Docker container的prometheus比較性能才公平



如何公平比較 Docker Compose 部署的 VictoriaMetrics Cluster 與單個 Docker 容器的 Prometheus 的性能

要公平比較 使用 `docker-compose` 部署的 VictoriaMetrics Cluster (vmstorage, vminsert、vmselect、vmagent、vmalert) 和 單個 Docker 容器的 Prometheus，需要考慮多個層面的性能指標。以下是詳細的比較方法：

1. 定義比較的性能指標 (KPI)

公平的性能比較應涵蓋以下幾個核心指標：

| KPI | 說明 | 測試方式 |
|---------|------------------------|--|
| CPU 使用率 | 監控各系統的 CPU 消耗 | `docker stats` 或 `process_cpu_seconds_total` |
| 記憶體 使用量 | 監控記憶體消耗 | `docker stats` 或 `process_resident_memory_bytes` |
| I/O 負載 | 監測寫入/讀取速率 | `node_disk_io_time_seconds_total` |
| 數據存儲壓縮率 | 比較存儲相同數據所需的磁碟空間 | `du -sh /var/lib/prometheus` (Prometheus) vs. `du -sh /storage` (VictoriaMetrics) |
| 數據查詢延遲 | 比較 PromQL 查詢相同指標時的響應時間 | `histogram_quantile(0.99, rate(http_request_duration_seconds_bucket[5m]))` |
| 數據寫入吞吐量 | 測試每秒可處理的 metrics 條數 | `rate(vm_inserted_rows_total[5m])` (VM) vs. `rate(prometheus_tsdb_head_samples_appended_total[5m])` (Prometheus) |





| KPI | 說明 | 測試方式 |
|----------|---------------------------------------|---|
| HTTP | 測試 | 使用 `curl` 或 `wrk` 工具進行測試 |
| API 響響時間 | `/api/v1/query_range` | |
| 應時間 | API 的延遲 | |
| 遠端存儲 | 比較兩者 | `rate(remote_write_requests_total[5m])` |
| 儲存速度 | `remote_write` 和 `remote_read` 的速度 | |

2. 測試環境統一

為了讓測試環境公平，需要確保：

- 相同的硬體環境 (CPU 核心數、記憶體大小、磁碟 IOPS)
- 相同的測試數據來源 (相同的 metrics 和寫入速率)
- 相同的監控工具 (使用 `Prometheus` 和 `Grafana` 來監控 VictoriaMetrics 和 Prometheus 自身)
- 相同的測試時間範圍 (例如，測試 1 小時、6 小時或 24 小時)

3. 部署測試環境

3.1 VictoriaMetrics Cluster (`docker-compose.yml`)

```
version: '3.7'
services:
  vmstorage:
    image: victoriametrics/vmstorage
    container_name: vmstorage
    restart: always
    volumes:
      - vmstorage-data:/storage
    networks:
      - monitoring
```





```
vminsert:
  image: victoriametrics/vminsert
  container_name: vminsert
  restart: always
  command: -storageNode=vmstorage:8400
  networks:
    - monitoring

vmselect:
  image: victoriametrics/vmselect
  container_name: vmselect
  restart: always
  command: -storageNode=vmstorage:8400
  networks:
    - monitoring

vmagent:
  image: victoriametrics/vmagent
  container_name: vmagent
  restart: always
  command: -remoteWrite.url=http://vminsert:8480/insert/
  networks:
    - monitoring

vmalert:
  image: victoriametrics/vmalert
  container_name: vmalert
  restart: always
  command: -datasource.url=http://vmselect:8481/select/0/prometheus
  networks:
    - monitoring

networks:
  monitoring:

volumes:
  vmstorage-data:
```

3.2 單容器 Prometheus (`docker-compose.yml`)

```
version: '3.7'
services:
```





```
prometheus:
  image: prom/prometheus
  container_name: prometheus
  restart: always
  volumes:
    - prometheus-data:/prometheus
  command:
    - --storage.tsdb.path=/prometheus
    - --storage.tsdb.retention.time=15d
    - --config.file=/etc/prometheus/prometheus.yml
  networks:
    - monitoring

networks:
  monitoring:

volumes:
  prometheus-data:
```

4. 執行測試

在兩個環境上執行相同的測試負載。

4.1 測試數據寫入性能

- 使用 `vmagent` 和 `remote_write` 來發送大量時序數據到 VictoriaMetrics 和 Prometheus。
- 使用 `stress-ng` 來模擬高負載的數據寫入：

```
stress-ng --cpu 4 --vm 2 --vm-bytes 1G --timeout 60s
```

- 觀察 `rate(vm_inserted_rows_total[5m])` 和 `rate(prometheus_tsdb_head_samples_appended_total[5m])` 來比較寫入速率。

4.2 測試查詢延遲





- 在 Prometheus UI 內測試查詢：

```
histogram_quantile(0.99,  
rate(http_request_duration_seconds_bucket[5m]))
```

- 使用 `wrk` 工具測試 HTTP API：

```
wrk -t12 -c400 -d30s "http://localhost:9090/api/v1/query?query=up"
```

- 比較 `rate(http_request_duration_seconds_bucket[5m])` 的響應時間。

4.3 測試 CPU 和記憶體使用

- 透過 `docker stats` 或 `PromQL` 查詢：

```
docker stats
```

或在 Prometheus UI 中查詢：

```
rate(process_cpu_seconds_total[5m])
```

4.4 測試磁碟壓縮率

- 在宿主機執行：

```
du -sh /var/lib/prometheus  
du -sh /storage
```

- 比較存儲相同數據時，VictoriaMetrics 和 Prometheus 所佔用的磁碟空間。

5. 解析測試結果

VictoriaMetrics

- 優勢：





- 寫入性能較高 (`vm_inserted_rows_total` 高於 `prometheus_tsdb_head_samples_appended_total`)
 - 查詢速度快 (`vmselect` 具備更快的索引)
 - 存儲壓縮率高 (磁碟佔用較小)
 - 支持分布式架構 (可水平擴展)
- 劣勢：
- 部署較複雜 (需要 `vmstorage`, `vminsert`, `vmselect`)
 - 需要調優不同組件

Prometheus

- 優勢：
 - 單一容器，易於部署
 - 與 Kubernetes / Thanos 集成更成熟
 - 生態系統廣泛
- 劣勢：
 - 寫入性能較低
 - 查詢延遲較高
 - 磁碟壓縮效率較低

6. 總結

| 指標 | VictoriaMetrics | Prometheus |
|--------|-----------------|------------|
| 寫入性能 | 高 | 中等 |
| 查詢速度 | 快 | 較慢 |
| 磁碟佔用 | 低 | 高 |
| 系統資源佔用 | 低 | 高 |
| 擴展性 | 水平擴展 | 需 Thanos |



如果你的監控數據量大、查詢需求頻繁，**VictoriaMetrics** 更適合；如果希望簡單部署、與 Kubernetes 整合，則 **Prometheus** 更方便。



以繁體中文詳細說明:2. 測試環境統一、相同的硬體環境（CPU 核心數、記憶體大小、磁碟 IOPS）



如何確保測試環境統一：相同的硬體環境

在比較 **VictoriaMetrics Cluster**（使用 `docker-compose` 部署）和 **單一 Docker 容器的 Prometheus** 性能時，確保兩者運行在相同的硬體環境中是 **公平測試的關鍵**。如果硬體資源不同，測試結果將無法準確反映系統效能差異。以下是詳細的測試環境統一指南。

1. 需要確保的硬體條件

公平比較時，應確保兩者的 **CPU、記憶體、磁碟 IOPS** 等資源 **相等**，具體包括：

| 參數 | 說明 | 測試方式 |
|--|--|---|
| CPU 核心數 (vCPU) | 確保 VictoriaMetrics 和 Prometheus 運行在相同的 CPU 配置上 | `lscpu` / `cat /proc/cpuinfo` |
| 記憶體大小 | 確保兩者可用的記憶體相等 | `free -h` / `docker stats` |
| 磁碟 IOPS (Input/Output Per Second) | 確保兩者使用相同的磁碟 IOPS | `fio` 測試 |
| 網絡帶寬 | 確保兩者的網絡流量受相同限制 | `iperf3` 測試 |
| Docker 資源限制 | 限制容器 CPU / Memory，避免資源佔用不均 | `docker run --cpus` / `docker run --memory` |
| 測試數據負載 | 保持相同的監控數據量、寫入頻率、查詢次數 | `stress-ng`、`wrk` |





2. 確保相同的 CPU 核心數

2.1 檢查主機 CPU

在測試環境中執行：

```
lscpu
```

輸出範例如：

```
CPU(s):          8
Thread(s) per core: 2
Core(s) per socket: 4
Socket(s):        1
```

確保 **CPU(s)** (總核心數) 在 VictoriaMetrics 和 Prometheus 測試環境中 相等。

2.2 限制 Docker 容器的 CPU 使用

在 `docker-compose.yml` 限制 **每個容器** 的 CPU 數量：

```
vminsert:
  deploy:
    resources:
      limits:
        cpus: "2"
```

對 **Prometheus** 容器 也進行相同的 CPU 限制：

```
prometheus:
  deploy:
    resources:
      limits:
        cpus: "2"
```

如果不使用 `docker-compose`，可以手動執行：



```
docker run --cpus="2" prom/prometheus
```

3. 確保相同的記憶體大小

3.1 檢查主機記憶體

在測試環境中執行：

```
free -h
```

輸出範例如：

| | total | used | free |
|-------|-------|------|------|
| Mem: | 16Gi | 2Gi | 14Gi |
| Swap: | 2Gi | 1Gi | 1Gi |

確保 Prometheus 和 VictoriaMetrics 運行的環境擁有 **相等的可用記憶體**。

3.2 限制 Docker 容器的記憶體

在 `docker-compose.yml` 設定 **記憶體限制**：

```
vminsert:  
  deploy:  
    resources:  
      limits:  
        memory: "4GB"
```

對 **Prometheus** 容器 設定相同的記憶體：

```
prometheus:  
  deploy:  
    resources:
```





```
limits:  
  memory: "4GB"
```

若直接運行容器：

```
docker run --memory="4g" prom/prometheus
```

4. 確保相同的磁碟 IOPS

VictoriaMetrics 和 Prometheus 都是時序數據庫，磁碟 I/O 影響存取效率，因此需要確保磁碟性能一致。

4.1 檢查磁碟 IOPS

使用 `fio` 測試讀寫性能：

```
fio --name=test --size=500M --rw=randrw --bs=4k --numjobs=4 --time_based -  
-runtime=30s --group_reporting
```

輸出範例如：

```
read: IOPS=12345  
write: IOPS=11234
```

如果兩個環境的 **IOPS** 相差過大，建議：

- 使用相同類型的 **SSD** 或 **HDD**
- 使用相同的 **檔案系統 (ext4、xfs 等)**
- **關閉 journaling** 減少磁碟寫入延遲

4.2 限制 Docker 容器的 I/O

使用 `--device-read-bps` 和 `--device-write-bps` 限制 I/O：





```
docker run --device-read-bps /dev/sda:100mb --device-write-bps  
/dev/sda:100mb prom/prometheus
```

或者在 `docker-compose.yml` :

```
prometheus:  
  deploy:  
    resources:  
      limits:  
        blkio_weight: 100
```

5. 確保相同的網絡帶寬

VictoriaMetrics 需要 `vmagent` 發送數據到 `vminsert`，可能會影響網絡流量，需統一 網絡頻寬。

5.1 測試網絡帶寬

使用 `iperf3` :

```
iperf3 -c <server-ip>
```

確保 VictoriaMetrics 和 Prometheus 測試環境 擁有相同的網絡頻寬。

5.2 限制 Docker 容器的網絡速率

使用 `tc` 限制容器網絡：

```
tc qdisc add dev eth0 root tbm rate 100mbit burst 32kbit latency 400ms
```

若使用 `docker-compose` :

```
prometheus:  
  networks:
```





```
default:  
  driver: bridge  
  driver_opts:  
    com.docker.network.bridge.default_bridge: "true"
```

6. 確保相同的測試數據

VictoriaMetrics 和 Prometheus 需要 **相同的監控數據** 來進行測試。可以透過 `vmagent` 或 `prometheus` 來產生相同的測試數據。

6.1 產生相同的測試數據

使用 `vmagent` 發送相同的 metrics：

```
vmagent -remoteWrite.url="http://localhost:8480/insert/" -  
promscrape.config="scrape_config.yaml"
```

或使用 `stress-ng` 產生高負載：

```
stress-ng --cpu 4 --vm 2 --vm-bytes 1G --timeout 60s
```

7. 確保相同的測試時間範圍

測試 VictoriaMetrics 和 Prometheus 的性能時，需確保測試執行時間一致：

- **短期測試** (30 分鐘)：觀察短時間內的負載表現
- **中期測試** (6 小時)：觀察長時間的 CPU、記憶體使用情況
- **長期測試** (24 小時+)：測試數據存儲、查詢效率、磁碟增長速度

可以使用 `timeout` 限制測試時間：



```
timeout 3600s stress-ng --cpu 4 --vm 2 --vm-bytes 1G
```

8. 總結

| 項目 | 設定方式 |
|---------|--|
| CPU | `docker run --cpus="2"` 或 `deploy.limits.cpus: "2"` |
| 記憶體 | `docker run --memory="4g"` 或 `deploy.limits.memory: "4GB"` |
| 磁碟 IOPS | `fio` 測試 + `docker run --device-read-bps` |
| 網絡頻寬 | `iperf3` 測試 + `tc qdisc` 限制 |
| 測試數據 | `vmagent` 或 `stress-ng` 產生相同數據 |
| 測試時間 | `timeout 3600s` 限制測試時間 |

這樣可以確保 **VictoriaMetrics** 和 **Prometheus** 在相同環境下進行公平測試，得到準確的性能比較結果！🚀



以繁體中文詳細說明:我要建立vmstorage vmselect vm insert
vmagent vmalert在同一台虛擬機上，請給我container mem的配置建議



在單台虛擬機上部署 VictoriaMetrics Cluster (**vmstorage**, **vmselect**, **vminsert**, **vmagent**, **vmalert**) 的記憶體 (Memory) 配置建議

VictoriaMetrics 是一款高效能、低資源消耗的時序數據庫 (TSDB)，適用於監控和遙測系統。在單台虛擬機上部署 **vmstorage**、**vmselect**、**vminsert**、**vmagent**、**vmalert** 時，為了確保系統的 **穩定性** 和 **效能**，我們需要合理配置每個容器的記憶體 (memory)。





1. 影響記憶體配置的因素

在分配記憶體時，需要考慮以下幾個因素：

1. 可用記憶體總量 (VM的 RAM 大小，如 16GB、32GB)
2. 每個組件的功能與記憶體需求
3. 流量負載 (每秒處理的 metrics 條數)
4. 查詢頻率 (PromQL 查詢的複雜度與並發量)
5. 歷史數據存儲需求
6. 預留給 Linux 和 Docker 其他進程的記憶體

VictoriaMetrics 官方建議 至少 **2GB** 記憶體 可運行一個基本的 VictoriaMetrics 服務，隨著負載增加，應適當調整。

2. 各組件的記憶體需求

| 組件 | 角色 | 記憶體需求 (最低) | 記憶體需求 (建議) |
|------------------|--------------------------|--------------|---------------|
| vmstorage | 負責存儲所有數據 | 2GB | 4GB+ |
| vminsert | 負責數據寫入 | 512MB | 1GB+ |
| vmselect | 負責查詢請求 | 1GB | 2GB+ |
| vmagent | 收集 metrics 並發送到 vminsert | 256MB | 512MB+ |
| vmalert | 負責告警 (Alerting) | 128MB | 256MB+ |
| 總計 | - | 約 4GB | 8GB 以上 |

“說明：”

- 若 `vmstorage` 的數據量較大，記憶體需求可能超過 **8GB**。
- `vmselect` 查詢時會消耗額外記憶體，若查詢壓力高，應給 **2GB 以上**。
- `vmagent` 主要負責收集和轉發 metrics，記憶體需求較低。





3. 記憶體配置建議

(1) 假設 VM 配置：16GB RAM

如果你的虛擬機擁有 **16GB RAM**，則可分配：

- `vmstorage` → **5GB**
- `vmselect` → **3GB**
- `vminsert` → **2GB**
- `vmagent` → **1GB**
- `vmalert` → **512MB**
- 預留 **4.5GB** 細 Linux 系統 & Docker 其他開銷

設定方式 (`docker-compose.yml`)

```
version: '3.7'
services:
  vmstorage:
    deploy:
      resources:
        limits:
          memory: "5GB"

  vmselect:
    deploy:
      resources:
        limits:
          memory: "3GB"

  vminsert:
    deploy:
      resources:
        limits:
          memory: "2GB"

  vmagent:
    deploy:
      resources:
        limits:
```





```
memory: "1GB"

vmalert:
  deploy:
    resources:
      limits:
        memory: "512MB"
```

(2) 假設 VM 配置：8GB RAM

如果你的虛擬機只有 **8GB RAM**，則需更嚴格限制：

- `vmstorage` → **3GB**
- `vmselect` → **2GB**
- `vminsert` → **1GB**
- `vmagent` → **512MB**
- `vmalert` → **256MB**
- **預留 1.5GB 紿 Linux 及 Docker**

設定方式 (docker-compose.yml)

```
version: '3.7'
services:
  vmstorage:
    deploy:
      resources:
        limits:
          memory: "3GB"

  vmselect:
    deploy:
      resources:
        limits:
          memory: "2GB"

  vminsert:
    deploy:
      resources:
```





```
limits:  
  memory: "1GB"  
  
vmagent:  
  deploy:  
    resources:  
      limits:  
        memory: "512MB"  
  
vmalert:  
  deploy:  
    resources:  
      limits:  
        memory: "256MB"
```

4. 額外優化建議

(1) 啟用 SWAP (避免 OOM)

VictoriaMetrics 可能會在高查詢負載時消耗額外記憶體，建議在 VM 上設定 SWAP 來防止 OOM (Out of Memory)。

在 Linux 上執行：

```
sudo fallocate -l 4G /swapfile  
sudo chmod 600 /swapfile  
sudo mkswap /swapfile  
sudo swapon /swapfile
```

並在 `/etc/fstab` 中加入：

```
/swapfile swap swap defaults 0 0
```

確保重啟後 SWAP 可用。





(2) 監控記憶體使用情況

可以使用 `docker stats` 來監控容器的記憶體使用情況：

```
docker stats
```

或透過 Prometheus 監控：

```
process_resident_memory_bytes{job="victoriameetrics"}
```

如果 `vmstorage` 或 `vmselect` 記憶體過高，可手動調整分配。

(3) 調整 VictoriaMetrics 配置

可透過環境變數來限制記憶體使用：

- **vmstorage :**

```
command:  
- "-memory.allowedPercent=50"
```

這會限制 VictoriaMetrics 只能使用總記憶體的 50%。

- **vmselect :**

```
command:  
- "-search.maxQueryLen=100000"
```

限制查詢長度，避免查詢過大導致記憶體消耗過多。

5. 總結





| VM 配置 | 16GB RAM 建議配置 | 8GB RAM 建議配置 |
|------------------|---------------|--------------|
| vmstorage | 5GB | 3GB |
| vmselect | 3GB | 2GB |
| vminsert | 2GB | 1GB |
| vmagent | 1GB | 512MB |
| vmalert | 512MB | 256MB |
| 系統預留 | 4.5GB | 1.5GB |

- 若 **查詢頻率高**，可提高 **vmselect** 記憶體。
- 若 **寫入數據量大**，可提高 **vmstorage** 記憶體。
- 可啟用 **SWAP**，避免記憶體不足導致 OOM。
- 使用 **Prometheus 監控** 記憶體使用，隨時調整。

這樣可以確保 **VictoriaMetrics Cluster** 在單台 VM 上穩定運行！



以繁體中文詳細說明:給我上述mem建議配置可參照的出處



在先前的討論中，我們針對在單台虛擬機上部署 **VictoriaMetrics Cluster**（包含 `vmstorage`、`vmselect`、`vminsert`、`vmagent`、`vmalert` 等組件）時的記憶體配置提出了建議。這些建議主要基於官方文檔和實踐經驗，旨在確保系統的穩定性和性能。以下是相關建議的參考來源：

1. 官方資源預留建議：

- 官方建議為所有實例類型預留 **50%** 的可用內存，以避免在負載高峰期間出現內存不足（OOM）或性能下降的情況。
- 同時，建議預留 **50%** 的空閒 CPU，以應對臨時的負載激增。
- 此外，對於 `vmstorage`，建議其數據存儲目錄至少預留 **20%** 的空閒磁碟空間。

VICTRIAMETRICS.COM.CN



2. 內存不足與性能問題的處理：

- 如果在數據寫入過程中出現緩慢的情況，可能是由於 `storage/tsid` 緩存未命中率較高。官方建議增加主機的可用內存，直到慢速寫入的比例低於 **5%**。
- 這可以通過增加每個現有 `vmstorage` 節點的可用內存，或向集群中添加更多的 `vmstorage` 節點來實現。

VICTORIAMETRICS.COM.CN

3. 資源利用率與系統穩定性：

- 為了確保系統在負載高峰期間的穩定性，建議在生產環境中保持以下資源使用率：
 - CPU 使用率**：保持在 **50%** 以下，以應對臨時的負載增加。
 - 內存使用率**：保持在 **50%** 以下，確保操作系統有足夠的內存用於頁面緩存，從而提高數據查詢和寫入的性能。
 - 磁碟空間**：確保至少有 **20%** 的可用磁碟空間，以便系統能夠執行必要的後台合併操作，維持數據寫入和查詢的效率。

VICTORIAMETRICS.COM.CN

這些建議旨在為您的部署提供一個穩定且高效的運行環境。實際配置應根據您的系統資源和業務需求進行調整。

資料來源



以繁體中文詳細說明:我要建立單點prometheus在同一台虛擬機上，請給我container mem的配置建議並給我給我上述mem建議配置可參照的出處



在單台虛擬機上部署 **Prometheus** 容器時，合理配置記憶體（Memory）限制有助於確保系統的穩定性和效能。以下是關於 Prometheus 容器記憶體配置的建議及其參考來源：





1. 記憶體配置建議

Prometheus 的記憶體使用量主要取決於以下因素：

- **監控指標的數量**：收集的 metrics 數量越多，佔用的記憶體越大。
- **數據保留時間**：數據保留時間越長，所需的存儲和記憶體資源越多。
- **查詢頻率和複雜度**：頻繁且複雜的查詢會增加記憶體的使用。

根據實踐經驗，對於中小型監控環境，建議為 Prometheus 容器分配 **2GB 至 4GB** 的記憶體。對於大型環境或高負載場景，可能需要分配更多的記憶體資源。

設定方式 (docker-compose.yml) :

```
version: '3.7'
services:
  prometheus:
    image: prom/prometheus
    container_name: prometheus
    restart: always
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
      - prometheus-data:/prometheus
    command:
      - --config.file=/etc/prometheus/prometheus.yml
      - --storage.tsdb.path=/prometheus
      - --storage.tsdb.retention.time=15d
    deploy:
      resources:
        limits:
          memory: 4GB
        reservations:
          memory: 2GB
    ports:
      - 9090:9090
volumes:
  prometheus-data:
```

說明 :





- **memory 限制**：上述配置將 Prometheus 容器的記憶體限制設置為 **4GB**，並預留 **2GB**。根據實際需求，可調整這些數值。
- **數據保留時間**：`--storage.tsdb.retention.time=15d` 指定數據保留時間為 15 天，可根據需求調整。

2. 參考來源

有關 Prometheus 容器部署和配置的詳細資訊，可參考以下資源：

- **Prometheus 官方文檔**：提供了關於 Prometheus 配置和資源管理的詳細說明。
- **CSDN 博客文章**：[《Prometheus 容器部署及應用》](#) 介紹了如何使用 Docker 部署 Prometheus 容器，並對配置進行了詳細說明。
- **博客園文章**：[《Docker 容器資源限制和監控》](#) 討論了如何限制和監控 Docker 容器的資源使用，包括記憶體限制的設置方法。

這些資源提供了關於 Prometheus 容器記憶體配置的詳細資訊，可供參考以確保系統的穩定運行。

資料來源



以繁體中文詳細說明:我要建立vmstorage vmselect vm insert
vmagent vmalert在同一台虛擬機上，請給我container cpu的配置建議



在單台虛擬機上部署 VictoriaMetrics Cluster (**vmstorage**, **vmselect**, **vminsert**, **vmagent**, **vmalert**) 的 CPU 配置建議

在單台虛擬機上部署 VictoriaMetrics Cluster，包括 **vmstorage**、**vmselect**、**vminsert**、**vmagent**、**vmalert**，需要合理配置 CPU 資源，以確保系統的高效運行，避





免單個組件佔用過多 CPU，影響整體效能。

1. 影響 CPU 配置的因素

在分配 CPU 時，需要考慮以下因素：

1. 虛擬機的 CPU 核心數 (如 4 核、8 核)
2. 監控數據寫入速率 (metrics ingestion rate)
3. 查詢負載 (PromQL 複雜度與並發查詢數)
4. 警報評估頻率 (alert evaluation frequency)
5. Docker 內部資源調度
6. Linux 內核與系統進程的 CPU 需求

2. 各組件的 CPU 需求

| 組件 | 角色 | CPU 需求 (最低) | CPU 需求 (建議) |
|------------------|--------------------------|---------------|--------------|
| vmstorage | 存儲所有時序數據 | 1 核 | 2-4 核 |
| vminsert | 負責數據寫入 | 0.5 核 | 1 核 |
| vmselect | 處理查詢請求 | 1 核 | 2-4 核 |
| vmagent | 收集 metrics 並發送到 vminsert | 0.5 核 | 1 核 |
| vmalert | 負責告警 | 0.25 核 | 0.5 核 |
| 總計 | - | 3.25 核 | 7-8 核 |

“說明：”

- `vmstorage` 和 `vmselect` 是 VictoriaMetrics 中最吃 CPU 的組件，因為 `vmstorage` 需要高效存儲數據，而 `vmselect` 需要執行 PromQL 查詢。
- `vminsert` 和 `vmagent` 的 CPU 需求較低，因為它們主要負責數據轉發。





- `vmalert` 負責執行告警規則，通常只需要較少的 CPU。

3. CPU 配置建議

(1) 假設 VM 配置：8 核

如果你的虛擬機擁有 **8 核 CPU**，則建議分配：

- `vmstorage` → **3 核**
- `vmselect` → **2 核**
- `vminsert` → **1 核**
- `vmagent` → **1 核**
- `vmalert` → **0.5 核**
- **預留 0.5 核 級 Linux 及 Docker 其他開銷**

設定方式 (**docker-compose.yml**)

```
version: '3.7'
services:
  vmstorage:
    deploy:
      resources:
        limits:
          cpus: "3"
  vmselect:
    deploy:
      resources:
        limits:
          cpus: "2"
  vminsert:
    deploy:
      resources:
        limits:
          cpus: "1"
  vmagent:
```





```
deploy:
  resources:
    limits:
      cpus: "1"

vmalert:
  deploy:
    resources:
      limits:
        cpus: "0.5"
```

(2) 假設 VM 配置：4 核心

如果你的虛擬機只有 **4 核 CPU**，則需要更精細的分配：

- `vmstorage` → **1.5 核**
- `vmselect` → **1 核**
- `vminsert` → **0.5 核**
- `vmagent` → **0.5 核**
- `vmalert` → **0.25 核**
- **預留 0.25 核 級 Linux 及 Docker 其他開銷**

設定方式 (**docker-compose.yml**)

```
version: '3.7'
services:
  vmstorage:
    deploy:
      resources:
        limits:
          cpus: "1.5"

  vmselect:
    deploy:
      resources:
        limits:
          cpus: "1"
```





```
vminsert:  
  deploy:  
    resources:  
      limits:  
        cpus: "0.5"  
  
vmagent:  
  deploy:  
    resources:  
      limits:  
        cpus: "0.5"  
  
vmalert:  
  deploy:  
    resources:  
      limits:  
        cpus: "0.25"
```

4. 額外優化建議

(1) 限制 CPU 過載

如果 `vmstorage` 或 `vmselect` 佔用過高 CPU，可以使用 **優先級調整 (nice 值)**：

```
vmstorage:  
  command:  
    - "--storageRetentions=30d"  
    - "--memory.allowedPercent=60"
```

這樣 `vmstorage` 只會使用 **60%** 的可用記憶體，降低 CPU 負擔。

(2) 監控 CPU 使用情況

可以使用 `docker stats` 監控 CPU：



```
docker stats
```

或者使用 Prometheus 監控：

```
rate(process_cpu_seconds_total{job="victoriametrics"}[5m])
```

如果 `vmstorage` 或 `vmselect` CPU 過高，可考慮調整 `cpus` 限制。

(3) 調整 VictoriaMetrics 配置

可透過環境變數來限制 CPU 使用：

- **vmstorage :**

```
command:  
- "-cpu.allowedPercent=75"
```

這會限制 VictoriaMetrics 只能使用 **75%** 的 CPU。

- **vmselect :**

```
command:  
- "-search.maxQueryLen=100000"
```

限制查詢長度，避免查詢過大導致 CPU 負載過高。

5. 總結

| VM 配置 | 8 核 CPU 建議配置 | 4 核 CPU 建議配置 |
|------------------|--------------|--------------|
| vmstorage | 3 核 | 1.5 核 |
| vmselect | 2 核 | 1 核 |





| VM 配置 | 8 核 CPU 建議配置 | 4 核 CPU 建議配置 |
|-----------------|--------------|--------------|
| vminsert | 1 核 | 0.5 核 |
| vmagent | 1 核 | 0.5 核 |
| vmalert | 0.5 核 | 0.25 核 |
| 系統預留 | 0.5 核 | 0.25 核 |

- 若 **查詢負載高**，可提高 **vmselect CPU**。
- 若 **寫入數據量大**，可提高 **vmstorage CPU**。
- 使用 **Prometheus 監控** CPU 使用情況，隨時調整。

這樣可以確保 **VictoriaMetrics Cluster** 在單台 VM 上穩定運行！



以繁體中文詳細說明:給我上述cpu建議配置可參照的出處



在先前的討論中，我們針對在單台虛擬機上部署 **VictoriaMetrics Cluster**（包括 `vmstorage`、`vmselect`、`vminsert`、`vmagent`、`vmalert` 等組件）時的 CPU 配置提出了建議。這些建議主要基於官方文檔和實踐經驗，旨在確保系統的穩定性和性能。以下是相關建議的參考來源：

1. 官方資源預留建議：

- 官方建議為所有實例類型預留 **50%** 的可用 CPU，以避免在負載高峰期間出現資源不足或性能下降的情況。
- 同時，建議預留 **50%** 的可用內存，以應對臨時的負載激增。
- 此外，對於 `vmstorage`，建議其數據存儲目錄至少預留 **20%** 的空閒磁碟空間。

VICTORIAMETRICS.COM.CN

2. 資源利用率與系統穩定性：

- 為了確保系統在負載高峰期間的穩定性，建議在生產環境中保持以下資源使用率：
 - **CPU 使用率**：保持在 **50%** 以下，以應對臨時的負載增加。





- **內存使用率**：保持在 **50%** 以下，確保操作系統有足夠的內存用於頁面緩存，從而提高數據查詢和寫入的性能。
- **磁碟空間**：確保至少有 **20%** 的可用磁碟空間，以便系統能夠執行必要的後台合併操作，維持數據寫入和查詢的效率。

VICTORIAMETRICS.COM.CN

這些建議旨在為您的部署提供一個穩定且高效的運行環境。實際配置應根據您的系統資源和業務需求進行調整。

資料來源



以繁體中文詳細說明:我要建立單點prometheus在同一台虛擬機上，
請給我container cpu的配置建議並給我給我上述cpu建議配置可參照
的出處



在單台虛擬機上部署 **Prometheus** 容器時，合理配置 CPU 資源有助於確保系統的穩定性和效能。以下是關於 Prometheus 容器 CPU 配置的建議及其參考來源：

1. CPU 配置建議

Prometheus 的 CPU 使用量主要取決於以下因素：

- **監控指標的數量**：收集的 metrics 越多，處理需求越高。
- **數據抓取頻率**：抓取頻率越高，CPU 負載越大。
- **查詢頻率和複雜度**：頻繁且複雜的查詢會增加 CPU 使用。

根據實踐經驗，對於中小型監控環境，建議為 Prometheus 容器分配 **1 至 2 個 CPU 核心**。對於大型環境或高負載場景，可能需要分配更多的 CPU 資源。

設定方式 (**docker-compose.yml**)：



Powered by [ChatGPT Exporter](#)



```
version: '3.7'
services:
  prometheus:
    image: prom/prometheus
    container_name: prometheus
    restart: always
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
      - prometheus-data:/prometheus
    command:
      - --config.file=/etc/prometheus/prometheus.yml
      - --storage.tsdb.path=/prometheus
      - --storage.tsdb.retention.time=15d
    deploy:
      resources:
        limits:
          cpus: "2.0"
        reservations:
          cpus: "1.0"
      ports:
        - 9090:9090
volumes:
  prometheus-data:
```

說明：

- **cpus 限制**：上述配置將 Prometheus 容器的 CPU 使用限制設置為 **2 個核心**，並預留 **1 個核心**。可根據實際需求調整這些數值。
- **數據保留時間**：`--storage.tsdb.retention.time=15d` 指定數據保留時間為 15 天，可根據需求調整。

2. 參考來源

有關 Docker 容器 CPU 資源限制的詳細資訊，可參考以下資源：

- **CSDN 博客文章**：[《如何设置Docker容器的CPU和内存使用限制》](#)介紹了如何通過 `docker run` 和 `docker-compose` 設置 Docker 容器的 CPU 及內存限制，包括設置



CPU 限制如使用兩個 CPU 核心，並解釋了 CPU 共享和內存保留的概念。

- **博客園文章：**[《Docker容器CPU、memory資源限制》](#) 討論了如何限制和監控 Docker 容器的資源使用，包括 CPU 和內存限制的設置方法。

這些資源提供了關於 Docker 容器 CPU 配置的詳細資訊，可供參考以確保系統的穩定運行。

資料來源



以繁體中文詳細說明:我用docker-compose 建立了vmstorage, vminsert, vmselect, vmagent, vmalert 各一個container，請問使用第三方監控工具vmagent + Grafana怎麼知道"一小時cpu、mem的使用率"、"寫入、查詢每筆資料的時間"、"一天儲存了幾筆用多少空間"



使用 **vmagent + Grafana** 監控 VictoriaMetrics Cluster

你已經使用 `docker-compose` 建立了 **vmstorage**、**vminsert**、**vmselect**、**vmagent**、**vmalert** 各一個 `container`，現在你希望透過 **vmagent + Grafana** 來監控：

1. 一小時 CPU、記憶體的使用率
2. 寫入、查詢每筆資料的時間
3. 一天內存儲的數據筆數與佔用的磁碟空間

步驟概述

1. 透過 `vmagent` 收集 VictoriaMetrics 各個組件的 metrics
2. 使用 PromQL 查詢 CPU、記憶體使用率
3. 分析寫入與查詢的時間
4. 計算一天內存儲的數據量與磁碟佔用空間
5. 在 Grafana 上建立對應的監控面板





1. 透過 `vmagent` 收集 VictoriaMetrics 監控數據

1.1 設定 `vmagent` 來收集 VictoriaMetrics Metrics

VictoriaMetrics 各組件都會暴露監控指標 (metrics)，可以透過 `vmagent` 來抓取並存入 `vminsert`。

建立 `scrape_config.yml`

在 `vmagent` 的 `scrape_config.yml` 中設定抓取目標：

```
global:
  scrape_interval: 15s  # 設定抓取頻率

scrape_configs:
  - job_name: "vmstorage"
    static_configs:
      - targets: ["vmstorage:8428"]

  - job_name: "vminsert"
    static_configs:
      - targets: ["vminsert:8480"]

  - job_name: "vmselect"
    static_configs:
      - targets: ["vmselect:8481"]

  - job_name: "vmagent"
    static_configs:
      - targets: ["vmagent:8429"]

  - job_name: "vmalert"
    static_configs:
      - targets: ["vmalert:8880"]
```

更新 `docker-compose.yml`

確保 `vmagent` 正確載入 `scrape_config.yml`：





```
vmagent:  
  image: victoriametrics/vmagent  
  container_name: vmagent  
  restart: always  
  volumes:  
    - ./scrape_config.yml:/etc/vmagent.yml  
  command:  
    - "-promscrape.config=/etc/vmagent.yml"  
    - "-remoteWrite.url=http://vminsert:8480/insert/"  
  ports:  
    - "8429:8429"
```

2. 監控 CPU、記憶體使用率

可以透過 `PromQL` 來查詢一小時內的 CPU 和記憶體使用率。

2.1 CPU 使用率

```
rate(process_cpu_seconds_total[1h])
```

- `process_cpu_seconds_total`：累積的 CPU 使用時間（秒）
- `rate(...[1h])`：計算過去 1 小時的 CPU 使用率

2.2 記憶體使用率

```
process_resident_memory_bytes
```

- `process_resident_memory_bytes`：當前記憶體使用量（bytes）
- 如果想查看一小時內的變化趨勢：

```
avg_over_time(process_resident_memory_bytes[1h])
```





3. 監控寫入與查詢時間

VictoriaMetrics 提供了一些指標來分析每筆資料的寫入與查詢時間。

3.1 監控寫入時間

```
rate(vm_insert_duration_seconds_sum[1h]) /  
rate(vm_insert_duration_seconds_count[1h])
```

- `vm_insert_duration_seconds_sum` : 累積的數據寫入時間
- `vm_insert_duration_seconds_count` : 寫入的總筆數
- **結果**：一筆數據的平均寫入時間（秒）

3.2 監控查詢時間

```
rate(vm_select_duration_seconds_sum[1h]) /  
rate(vm_select_duration_seconds_count[1h])
```

- `vm_select_duration_seconds_sum` : 累積的查詢時間
- `vm_select_duration_seconds_count` : 查詢的總次數
- **結果**：一筆查詢的平均時間（秒）

4. 監控一天內存儲的數據量與磁碟空間

4.1 監控一天內存儲的數據筆數

```
increase(vm_rows_inserted_total[1d])
```

- `vm_rows_inserted_total` : 累積的寫入行數
- `increase(...[1d])` : 統計一天內寫入的行數

4.2 監控磁碟佔用量





vm_data_size_bytes

- `vm_data_size_bytes` : VictoriaMetrics 佔用的磁碟空間 (bytes)

4.3 計算平均每筆數據的磁碟佔用

```
increase(vm_data_size_bytes[1d]) / increase(vm_rows_inserted_total[1d])
```

- 結果：每筆數據佔用的磁碟空間 (bytes/行)

5. 在 Grafana 設定監控面板

5.1 連接 Prometheus 數據源

1. 在 `Grafana` 介面 → `Configuration` → `Data Sources`
2. 新增 `Prometheus`
3. 設定 `URL` 為 `http://vmselect:8481/select/0/prometheus`
4. 點擊 `Save & Test`

5.2 建立監控面板

1. 建立 Dashboard

2. 新增 Panel

3. 輸入 PromQL 查詢

- CPU 使用率：`rate(process_cpu_seconds_total[1h])`
- 記憶體使用率：`process_resident_memory_bytes`
- 寫入平均時間：`rate(vm_insert_duration_seconds_sum[1h]) / rate(vm_insert_duration_seconds_count[1h])`
- 查詢平均時間：`rate(vm_select_duration_seconds_sum[1h]) / rate(vm_select_duration_seconds_count[1h])`





- 一天存儲筆數：`increase(vm_rows_inserted_total[1d])`
- 磁碟使用量：`vm_data_size_bytes`

4. 設定時間範圍

- CPU/記憶體使用率 → `Last 1h`
- 寫入/查詢時間 → `Last 1h`
- 一天內存儲筆數 → `Last 1d`
- 磁碟使用量 → `Last 1d`

5. 調整視覺化類型

- CPU/記憶體 → `Graph`
- 寫入/查詢時間 → `Bar Gauge`
- 存儲數據與磁碟佔用 → `SingleStat`

6. 點擊 Save，儲存 Dashboard

6. 總結

| 監控項目 | PromQL 查詢 | 說明 |
|---------|---|------------------|
| CPU 使用率 | `rate(process_cpu_seconds_total[1h])` | 計算 1 小時內 CPU 使用率 |
| 記憶體使用率 | `process_resident_memory_bytes` | 查看當前記憶體使用量 |
| 寫入時間 | `rate(vm_insert_duration_seconds_sum[1h]) / rate(vm_insert_duration_seconds_count[1h])` | 計算每筆數據的平均寫入時間 |
| 查詢時間 | `rate(vm_select_duration_seconds_sum[1h]) / rate(vm_select_duration_seconds_count[1h])` | 計算每筆查詢的平均時間 |
| 一天內存儲筆數 | `increase(vm_rows_inserted_total[1d])` | 計算一天內存儲的數據筆數 |
| 磁碟使用量 | `vm_data_size_bytes` | 計算磁碟總佔用空間 |





| 監控項目 | PromQL 查詢 | 說明 |
|--------------|--|------------------|
| 每筆數據 磁碟佔用 | `increase(vm_data_size_bytes[1d]) / increase(vm_rows_inserted_total[1d])` | 計算每筆數據的 磁碟使用量 |

這樣你就可以透過 **vmagent + Grafana** 監控 **VictoriaMetrics Cluster** 的 CPU、記憶體、寫入/查詢時間，以及一天內存儲的數據量和磁碟空間！🚀



以繁體中文詳細說明:我用docker 建立了promtheus的一個 container，請問使用第三方監控工具 + Grafana怎麼知道"一小時 cpu、mem的使用率"、"寫入、查詢每筆資料的時間"、"一天儲存了幾筆 用多少空間"



使用 Prometheus + Grafana 監控 Prometheus 本身的運行狀態

你已經使用 **Docker** 建立了一個 **Prometheus** 容器，現在希望透過 **第三方監控工具 + Grafana** 來監測：

1. 一小時 CPU、記憶體使用率
2. 寫入、查詢每筆資料的時間
3. 一天內存儲的數據筆數與佔用的磁碟空間

步驟概述

1. 透過 `node_exporter` 或 `cAdvisor` 收集 Prometheus 的 CPU、記憶體使用率
2. 使用 PromQL 查詢 CPU、記憶體使用狀況
3. 監控 Prometheus 寫入與查詢時間
4. 計算一天內存儲的數據量與磁碟佔用空間
5. 在 Grafana 上建立對應的監控面板





1. 安裝監控工具

為了監測 **Prometheus** 容器的資源使用狀況，我們需要部署 **cAdvisor** 或 **node_exporter** 來收集 Docker 資源數據。

1.1 更新 `docker-compose.yml`

在 `docker-compose.yml` 添加 **cAdvisor** 和 **Grafana**：

```
version: '3.7'
services:
  prometheus:
    image: prom/prometheus
    container_name: prometheus
    restart: always
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
      - prometheus-data:/prometheus
    command:
      - --config.file=/etc/prometheus/prometheus.yml
      - --storage.tsdb.path=/prometheus
      - --storage.tsdb.retention.time=15d
    ports:
      - 9090:9090

  cadvisor:
    image: gcr.io/cadvisor/cadvisor
    container_name: cadvisor
    restart: always
    ports:
      - "8080:8080"
    volumes:
      - "/:/rootfs:ro"
      - "/var/run:/var/run:rw"
      - "/sys:/sys:ro"
      - "/var/lib/docker/:/var/lib/docker:ro"

  grafana:
    image: grafana/grafana
    container_name: grafana
    restart: always
    ports:
```





```
- "3000:3000"
environment:
  - GF_SECURITY_ADMIN_PASSWORD=admin

volumes:
  prometheus-data:
```

1.2 配置 Prometheus 來抓取 `cAdvisor` 數據

在 `prometheus.yml` 配置文件中新增：

```
scrape_configs:
  - job_name: 'cadvisor'
    static_configs:
      - targets: ['cadvisor:8080']
```

然後 重啟 Prometheus：

```
docker-compose down && docker-compose up -d
```

2. 監控 CPU、記憶體使用率

可以透過 **PromQL** 來查詢 **1 小時內 Prometheus 的 CPU 和記憶體使用率**。

2.1 CPU 使用率

```
rate(process_cpu_seconds_total[1h])
```

- `process_cpu_seconds_total`：Prometheus 進程累積的 CPU 使用時間（秒）
- `rate(...[1h])`：計算過去 1 小時的 CPU 使用率

2.2 記憶體使用率





```
process_resident_memory_bytes
```

- `process_resident_memory_bytes`：當前記憶體使用量 (bytes)

2.3 Docker 容器的 CPU/記憶體使用率

如果使用 **cAdvisor** 監控 Prometheus 容器的資源：

```
rate(container_cpu_usage_seconds_total{name="prometheus"}[1h])
```

```
container_memory_usage_bytes{name="prometheus"}
```

- `container_cpu_usage_seconds_total`：Docker 容器的 CPU 使用時間
- `container_memory_usage_bytes`：Docker 容器的記憶體佔用量

3. 監控 Prometheus 寫入與查詢時間

Prometheus 提供了一些內建指標來監測 **每筆資料的寫入與查詢時間**。

3.1 監控寫入時間

```
rate(prometheus_tsdb_head_samples_appended_total[1h])
```

- `prometheus_tsdb_head_samples_appended_total`：每秒寫入的數據筆數

3.2 監控查詢時間

```
rate(prometheus_http_request_duration_seconds_sum{handler="/api/v1/query"}[1h]) /  
rate(prometheus_http_request_duration_seconds_count{handler="/api/v1/query"}[1h])
```





- `prometheus_http_request_duration_seconds_sum` : 累積查詢時間
 - `prometheus_http_request_duration_seconds_count` : 查詢請求數量
 - 結果：一筆查詢的平均時間（秒）
-

4. 監控一天內存儲的數據量與磁碟空間

4.1 監控一天內存儲的數據筆數

```
increase(prometheus_tsdb_head_samples_appended_total[1d])
```

- `prometheus_tsdb_head_samples_appended_total` : 累積的寫入行數
- `increase(...[1d])` : 統計一天內寫入的行數

4.2 監控磁碟佔用量

```
prometheus_tsdb_storage_blocks_bytes
```

- `prometheus_tsdb_storage_blocks_bytes` : Prometheus 佔用的磁碟空間 (bytes)

4.3 計算平均每筆數據的磁碟佔用

```
increase(prometheus_tsdb_storage_blocks_bytes[1d]) /  
increase(prometheus_tsdb_head_samples_appended_total[1d])
```

- 結果：每筆數據佔用的磁碟空間 (bytes/行)
-

5. 在 Grafana 設定監控面板

5.1 連接 Prometheus 數據源

1. 在 Grafana 介面 → `Configuration` → `Data Sources`





2. 新增 `Prometheus`
3. 設定 `URL` 為 `http://prometheus:9090`
4. 點擊 `Save & Test`

5.2 建立監控面板

1. 建立 Dashboard
2. 新增 Panel
3. 輸入 PromQL 查詢
 - CPU 使用率 : `rate(process_cpu_seconds_total[1h])`
 - 記憶體使用率 : `process_resident_memory_bytes`
 - 寫入平均時間 : `rate(prometheus_tsdb_head_samples_appended_total[1h])`
 - 查詢平均時間 :

```
`rate(prometheus_http_request_duration_seconds_sum{handler="/api/v1/query"}[1h]) / rate(prometheus_http_request_duration_seconds_count{handler="/api/v1/query"}[1h])`
```
 - 一天存儲筆數 :

```
`increase(prometheus_tsdb_head_samples_appended_total[1d])`
```
 - 磁碟使用量 : `prometheus_tsdb_storage_blocks_bytes`
4. 設定時間範圍
 - CPU/記憶體使用率 → `Last 1h`
 - 寫入/查詢時間 → `Last 1h`
 - 一天內存儲筆數 → `Last 1d`
 - 磁碟使用量 → `Last 1d`
5. 調整視覺化類型
 - CPU/記憶體 → `Graph`
 - 寫入/查詢時間 → `Bar Gauge`
 - 存儲數據與磁碟佔用 → `SingleStat`





6. 點擊 Save，儲存 Dashboard

這樣你就可以透過 **cAdvisor + Prometheus + Grafana** 監控 **Prometheus** 本身的 CPU、記憶體、寫入/查詢時間，以及一天內存儲的數據量和磁碟空間！ 



以繁體中文詳細說明:我用docker 建立了promtheus的一個 container，請問使用第三方監控工具prometheus + Grafana怎麼知道"一小時cpu、mem的使用率"、"寫入、查詢每筆資料的時間"、"一天儲存了幾筆 用多少空間"



使用 **Prometheus + Grafana** 監控 **Prometheus** 本身的運行狀態

你已經使用 **Docker** 建立了一個 **Prometheus** 容器，現在希望透過 **Prometheus + Grafana** 來監測：

1. 一小時 CPU、記憶體使用率
2. 寫入、查詢每筆資料的時間
3. 一天內存儲的數據筆數與佔用的磁碟空間

1. 設定 **Prometheus** 來監控自身

Prometheus 內建了 `'/metrics` 端點，可以提供自身運行的指標。首先，確保 Prometheus 監控自身。

1.1 更新 `prometheus.yml` 配置

```
global:  
  scrape_interval: 15s  # 每 15 秒抓取一次數據
```





```
scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']
```

1.2 確保 Prometheus 正確啟動

如果你是透過 `docker run` 啟動：

```
docker run -d --name prometheus -p 9090:9090 \
-v $(pwd)/prometheus.yml:/etc/prometheus/prometheus.yml \
prom/prometheus
```

如果你是透過 `docker-compose`：

```
version: '3.7'
services:
  prometheus:
    image: prom/prometheus
    container_name: prometheus
    restart: always
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
      - prometheus-data:/prometheus
    command:
      - --config.file=/etc/prometheus/prometheus.yml
      - --storage.tsdb.path=/prometheus
      - --storage.tsdb.retention.time=15d
    ports:
      - "9090:9090"
volumes:
  prometheus-data:
```

然後重啟：

```
docker-compose down && docker-compose up -d
```





2. 監控 CPU、記憶體使用率

2.1 監控 Prometheus 自身的 CPU 使用率

```
rate(process_cpu_seconds_total[1h])
```

- `process_cpu_seconds_total` : Prometheus 進程累積的 CPU 使用時間 (秒)
- `rate(...[1h])` : 計算過去 1 小時的 CPU 使用率

2.2 監控 Prometheus 自身的記憶體使用

```
process_resident_memory_bytes
```

- `process_resident_memory_bytes` : 當前記憶體使用量 (bytes)

3. 監控 Prometheus 寫入與查詢時間

Prometheus 提供了一些內建指標來監測 **每筆資料的寫入與查詢時間**。

3.1 監控每筆資料的寫入時間

```
rate(prometheus_tsdb_head_samples_appended_total[1h])
```

- `prometheus_tsdb_head_samples_appended_total` : 每秒寫入的數據筆數
- `rate(...[1h])` : 計算過去 1 小時的寫入速率

3.2 監控每筆查詢的時間

```
rate(prometheus_http_request_duration_seconds_sum{handler="/api/v1/query"}  
[1h])  
/
```





```
rate(prometheus_http_request_duration_seconds_count{handler="/api/v1/query"}[1h])
```

- `prometheus_http_request_duration_seconds_sum` : 累積查詢時間
- `prometheus_http_request_duration_seconds_count` : 查詢請求數量
- **結果**：一筆查詢的平均時間（秒）

4. 監控一天內存儲的數據量與磁碟空間

4.1 監控一天內存儲的數據筆數

```
increase(prometheus_tsdb_head_samples_appended_total[1d])
```

- `prometheus_tsdb_head_samples_appended_total` : 累積的寫入行數
- `increase(...[1d])` : 統計一天內寫入的行數

4.2 監控 Prometheus 佔用的磁碟空間

```
prometheus_tsdb_storage_blocks_bytes
```

- `prometheus_tsdb_storage_blocks_bytes` : Prometheus 佔用的磁碟空間 (bytes)

4.3 計算平均每筆數據的磁碟佔用

```
increase(prometheus_tsdb_storage_blocks_bytes[1d]) /  
increase(prometheus_tsdb_head_samples_appended_total[1d])
```

- **結果**：每筆數據佔用的磁碟空間 (bytes/行)

5. 在 Grafana 設定監控面板





5.1 連接 Prometheus 數據源

1. 在 **Grafana** 介面 → `Configuration` → `Data Sources`
2. 新增 `Prometheus`
3. 設定 `URL` 為 `http://prometheus:9090`
4. 點擊 `Save & Test`

5.2 建立監控面板

1. 建立 Dashboard

2. 新增 Panel

3. 輸入 PromQL 查詢

- **CPU 使用率** : `rate(process_cpu_seconds_total[1h])`
- **記憶體使用率** : `process_resident_memory_bytes`
- **寫入平均時間** : `rate(prometheus_tsdb_head_samples_appended_total[1h])`
- **查詢平均時間** :

```
`rate(prometheus_http_request_duration_seconds_sum{handler="/api/v1/query"}[1h]) / rate(prometheus_http_request_duration_seconds_count{handler="/api/v1/query"}[1h])`
```
- **一天存儲筆數** :

```
`increase(prometheus_tsdb_head_samples_appended_total[1d])`
```
- **磁碟使用量** : `prometheus_tsdb_storage_blocks_bytes`

4. 設定時間範圍

- **CPU/記憶體使用率** → `Last 1h`
- **寫入/查詢時間** → `Last 1h`
- **一天內存儲筆數** → `Last 1d`
- **磁碟使用量** → `Last 1d`

5. 調整視覺化類型

- **CPU/記憶體** → `Graph`





- 寫入/查詢時間 → `Bar Gauge`
- 存儲數據與磁碟佔用 → `SingleStat`

6. 點擊 Save, 儲存 Dashboard

6. 總結

監控

| 項目 | PromQL 查詢 | 說明 |
|---------|---|------------------|
| CPU 使用率 | `rate(process_cpu_seconds_total[1h])` | 計算 1 小時內 CPU 使用率 |
| 記憶體使用率 | `process_resident_memory_bytes` | 查看當前記憶體使用量 |
| 寫入時間 | `rate(prometheus_tsdb_head_samples_appended_total[1h])` | 計算每筆數據的寫入速率 |
| 查詢時間 | `rate(prometheus_http_request_duration_seconds_sum{handler="/api/v1/query"} [1h]) / rate(prometheus_http_request_duration_seconds_count{handler="/api/v1/query"} [1h])` | 計算每筆查詢的平均時間 |
| 一天內存儲筆數 | `increase(prometheus_tsdb_head_samples_appended_total[1d])` | 計算一天內存儲的 |





監控

項目 PromQL 查詢

| 項目 | PromQL 查詢 | 說明 |
|----------------------|--|----------------------------------|
| | | 數據 筆數 |
| 磁碟 使用 量 | `prometheus_tsdb_storage_blocks_bytes` | 計算 磁碟 總佔 用空 間 |
| 每筆 數據 磁碟 佔用 | `increase(prometheus_tsdb_storage_blocks_bytes[1d]) / increase(prometheus_tsdb_head_samples_appended_total[1d])` | 計算 每筆 數據 的磁 碟使 用量 |

這樣你就可以透過 **Prometheus + Grafana** 監控 **Prometheus** 本身的 CPU、記憶體、寫入/查詢時間，以及一天內存儲的數據量和磁碟空間！



以繁體中文詳細說明:請給我上述docker-compose 建立了
`vmstorage`、`vminsert`、`vmselect`、`vmagent`、`vmalert` 各一個
container、`prometheus`的一個container，對於"一小時cpu、
mem的使用率"、"寫入、查詢每筆資料的時間"、"一天儲存了幾筆
用多少空間"的promql的詳細說明，並附上可參照的出處



您已使用 Docker Compose 建立了以下容器：`vmstorage`、`vminsert`、`vmselect`、
`vmagent`、`vmalert`（VictoriaMetrics 集群）以及一個 `prometheus` 容器。現在，您希
望透過 Prometheus 和 Grafana 監控這些服務，特別關注以下指標：

1. 一小時內的 CPU 和記憶體使用率
2. 每筆資料的寫入和查詢時間
3. 每日存儲的數據量及所佔用的磁碟空間

以下是詳細的說明，包括相關的 PromQL 查詢語句和參考資料。





1. 監控一小時內的 CPU 和記憶體使用率

1.1 VictoriaMetrics 組件

VictoriaMetrics 的各個組件（如 `vmstorage`、`vminsert`、`vmselect` 等）會在 `/metrics` 端點暴露自身的指標。您可以配置 Prometheus 抓取這些指標，並使用以下 PromQL 查詢：

CPU 使用率：

```
rate(process_cpu_seconds_total{job="vm_component"}[1h])
```

- `process_cpu_seconds_total`：累積的 CPU 使用時間（秒）
- `job="vm_component"`：對應的 VictoriaMetrics 組件名稱（如 `vmstorage`、`vminsert` 等）
- `rate(...[1h])`：計算過去一小時的平均 CPU 使用率

記憶體使用量：

```
process_resident_memory_bytes{job="vm_component"}
```

- `process_resident_memory_bytes`：當前的記憶體使用量（位元組）
- `job="vm_component"`：對應的 VictoriaMetrics 組件名稱

1.2 Prometheus 容器

Prometheus 也會在 `/metrics` 端點暴露自身的指標。使用以下 PromQL 查詢來監控其資源使用情況：

CPU 使用率：

```
rate(process_cpu_seconds_total{job="prometheus"}[1h])
```

記憶體使用量：





```
process_resident_memory_bytes{job="prometheus"}
```

2. 監控每筆資料的寫入和查詢時間

2.1 VictoriaMetrics 組件

寫入時間：

VictoriaMetrics 的 `vminsert` 組件負責數據的寫入。您可以監控其寫入延遲：

```
rate(vm_insert_duration_seconds_sum[1h]) /  
rate(vm_insert_duration_seconds_count[1h])
```

- `vm_insert_duration_seconds_sum`：累積的寫入時間總和
- `vm_insert_duration_seconds_count`：寫入操作的總次數
- 上述查詢計算過去一小時內每次寫入操作的平均時間（秒）

查詢時間：

`vmselect` 組件負責數據的查詢。監控其查詢延遲：

```
rate(vm_select_duration_seconds_sum[1h]) /  
rate(vm_select_duration_seconds_count[1h])
```

- `vm_select_duration_seconds_sum`：累積的查詢時間總和
- `vm_select_duration_seconds_count`：查詢操作的總次數
- 上述查詢計算過去一小時內每次查詢操作的平均時間（秒）

2.2 Prometheus 容器

寫入時間：

Prometheus 的寫入操作主要涉及數據的抓取和存儲。可以監控以下指標：





```
rate(prometheus_tsdb_head_samples_appended_total[1h])
```

- `prometheus_tsdb_head_samples_appended_total`：每秒寫入的樣本數
- 此查詢顯示過去一小時內每秒寫入的樣本數量

查詢時間：

```
rate(prometheus_http_request_duration_seconds_sum{handler=~"/api/v1/query.*"}[1h]) /  
rate(prometheus_http_request_duration_seconds_count{handler=~"/api/v1/query.*"}[1h])
```

- `prometheus_http_request_duration_seconds_sum`：累積的 HTTP 查詢請求時間總和
- `prometheus_http_request_duration_seconds_count`：HTTP 查詢請求的總次數
- 上述查詢計算過去一小時內每次查詢請求的平均時間（秒）

3. 監控每日存儲的數據量及所佔用的磁碟空間

3.1 VictoriaMetrics 組件

每日存儲的數據筆數：

```
increase(vm_rows_inserted_total[1d])
```

- `vm_rows_inserted_total`：累積的插入行數
- `increase(...[1d])`：計算過去一天內插入的行數

磁碟空間使用量：

```
vm_data_size_bytes
```

- `vm_data_size_bytes`：當前的數據存儲大小（位元組）





3.2 Prometheus 容器

每日存儲的數據筆數：

```
increase(prometheus_tsdb_head_samples_appended_total[1d])
```

- `prometheus_tsdb_head_samples_appended_total`：累積的寫

資料來源



以繁體中文詳細說明:我用vmagent監控vmstorage、vminsert、vmselect、vmagent, scrape怎麼設定



如何使用 `vmagent` 監控 `vmstorage`、`vminsert`、`vmselect`、`vmagent` 的 `scrape` 設定

`vmagent` 是 VictoriaMetrics 提供的一個高效能 `scraper`，類似於 Prometheus 的 `scrape` 機制。當 `vmagent` 被配置為監控 `vmstorage`、`vminsert`、`vmselect`、`vmagent` 時，它會定期抓取這些組件的 `/metrics` 端點，並將數據傳送到 `vminsert` 進行存儲。

1. `scrape_config` 設定方式

VictoriaMetrics 使用 YAML 格式來配置 `vmagent`，類似於 Prometheus 的 `scrape_configs`。

1.1 建立 `scrape_config.yml`

在 `vmagent` 所在的伺服器或容器中，建立 `scrape_config.yml`，內容如下：





```
global:
  scrape_interval: 15s  # 每 15 秒抓取一次數據
  evaluation_interval: 15s # 每 15 秒評估一次規則

scrape_configs:
  - job_name: "vmstorage"
    static_configs:
      - targets: ["vmstorage:8428"]

  - job_name: "vminsert"
    static_configs:
      - targets: ["vminsert:8480"]

  - job_name: "vmselect"
    static_configs:
      - targets: ["vmselect:8481"]

  - job_name: "vmagent"
    static_configs:
      - targets: ["vmagent:8429"]
```

1.2 設定解析

- `scrape_interval: 15s` → 表示 **每 15 秒抓取一次 metrics**, 可根據需求調整 (預設值為 10s)。
- `evaluation_interval: 15s` → 表示 **每 15 秒評估一次規則** (如 `vmalert` 告警)。
- `job_name: "vmstorage"` → 定義 `vmstorage` 這個監控任務。
- `targets: ["vmstorage:8428"]` → `vmstorage` 的 `/metrics` 端點, 抓取監控指標。

2. `docker-compose.yml` 配置

確保 `vmagent` 容器啟動時能夠正確讀取 `scrape_config.yml`, 需要在 `docker-compose.yml` 中掛載這個檔案。





```
version: '3.7'
services:
  vmagent:
    image: victoriametrics/vmagent
    container_name: vmagent
    restart: always
    volumes:
      - ./scrape_config.yml:/etc/vmagent.yml
    command:
      - "-promscrape.config=/etc/vmagent.yml"
      - "-remoteWrite.url=http://vminsert:8480/insert/"
    ports:
      - "8429:8429"
    networks:
      - monitoring

  vmstorage:
    image: victoriametrics/vmstorage
    container_name: vmstorage
    restart: always
    ports:
      - "8428:8428"
    networks:
      - monitoring

  vminsert:
    image: victoriametrics/vminsert
    container_name: vminsert
    restart: always
    command: -storageNode=vmstorage:8400
    ports:
      - "8480:8480"
    networks:
      - monitoring

  vmselect:
    image: victoriametrics/vmselect
    container_name: vmselect
    restart: always
    command: -storageNode=vmstorage:8400
    ports:
      - "8481:8481"
    networks:
```





```
- monitoring
```

```
networks:
```

```
  monitoring:
```

2.1 設定解析

- `--promscrape.config=/etc/vmagent.yml` → 指定 `vmagent` 讀取 `scrape_config.yml` 配置文件。
- `--remoteWrite.url=http://vminsert:8480/insert/` → 指定 `vmagent` 將抓取的數據寫入 `vminsert`。
- `volumes:` → 掛載 `scrape_config.yml` 到 `vmagent` 容器內，確保其可以讀取 `scrape` 設定。

3. 測試 `vmagent` 是否正常運作

啟動 `docker-compose`：

```
docker-compose up -d
```

然後檢查 `vmagent` 是否正常運行：

```
docker logs -f vmagent
```

若 `vmagent` 正確抓取 `vmstorage`、`vminsert`、`vmselect`、`vmagent` 的 `metrics`，則會顯示類似以下的日誌：

```
scraping target="vmstorage:8428" success
scraping target="vminsert:8480" success
scraping target="vmselect:8481" success
scraping target="vmagent:8429" success
```





4. 如何在 Grafana 設置監控面板

4.1 連接 VictoriaMetrics 為資料來源

1. 打開 Grafana
2. 點擊 `Configuration` → `Data Sources`
3. 新增 `Prometheus`
4. 設定 `URL` 為：

```
http://vmselect:8481/select/0/prometheus
```

5. 點擊 `Save & Test`，如果成功，表示 `Grafana` 可以讀取 `vmagent` 提供的數據。

5. 監控 `vmstorage`、`vminsert`、`vmselect`、`vmagent` 指標

你可以在 Grafana 建立面板，並使用以下 **PromQL** 查詢來監控 VictoriaMetrics 各個組件的狀態。

5.1 監控 CPU 使用率

```
rate(process_cpu_seconds_total{job="vmstorage"}[1h])
```

- 計算 `vmstorage` 在 1 小時內的 CPU 使用率。

```
rate(process_cpu_seconds_total{job="vminsert"}[1h])
```

- 計算 `vminsert` 的 CPU 使用率。

5.2 監控記憶體使用量

```
process_resident_memory_bytes{job="vmstorage"}
```

- 顯示 `vmstorage` 的當前記憶體使用量 (bytes)。



```
process_resident_memory_bytes{job="vmselect"}
```

- 顯示 `vmselect` 的記憶體使用量。

5.3 監控寫入速度

```
rate(vm_insert_duration_seconds_sum[1h]) /  
rate(vm_insert_duration_seconds_count[1h])
```

- 計算 每筆數據的平均寫入時間 (秒)。

5.4 監控查詢速度

```
rate(vm_select_duration_seconds_sum[1h]) /  
rate(vm_select_duration_seconds_count[1h])
```

- 計算 每筆查詢的平均時間 (秒)。

5.5 監控存儲使用量

```
vm_data_size_bytes
```

- 獲取 VictoriaMetrics 當前存儲的數據大小 (bytes)。

```
increase(vm_rows_inserted_total[1d])
```

- 計算 過去 1 天內存儲的數據筆數。

6. 總結

- 配置 `vmagent` 的 `scrape_config.yml`，讓它監控 `vmstorage`、`vminsert`、`vmselect`、`vmagent`





2. 將 `vmagent` 寫入的數據發送到 `vminsert`
3. 使用 `PromQL` 查詢 CPU、記憶體、寫入速度、查詢時間、存儲使用量
4. 在 `Grafana` 設定 VictoriaMetrics 作為數據來源，並建立監控面板

這樣你就可以透過 `vmagent` 監控 `vmstorage`、`vminsert`、`vmselect`、`vmagent`，並在 `Grafana` 上查看 VictoriaMetrics 的性能狀態！



以繁體中文詳細說明:static_configs:
- targets: ["vmstorage:8428"]



`static_configs: - targets: ["vmstorage:8428"]` 詳細說明

在 `vmagent`（或 Prometheus）的 `scrape_config.yml` 中，這一行設定了 `vmagent` 定期抓取 `vmstorage` 服務的 `/metrics` 端點，並將收集到的監控數據傳送到指定的存儲端（如 `vminsert`）。

1. `static_configs` 的作用

`static_configs` 是 Prometheus 和 vmagent 的靜態目標設定，用來定義 監控哪些伺服器，以及它們的 metrics 端口。

```
scrape_configs:  
  - job_name: "vmstorage"  
    static_configs:  
      - targets: ["vmstorage:8428"]
```

這段設定的作用是：

1. `job_name: "vmstorage"`





- 定義 **監控任務的名稱**，這個名稱會顯示在 Prometheus/Grafana 中。

2. `static_configs`

- 使用 **靜態配置** 方式來指定要監控的服務 (``vmstorage``).

3. `targets: ["vmstorage:8428"]`

- 指定 **vmagent** 要抓取 `vmstorage` 指標的 HTTP 端點。
- `"vmstorage:8428"` 代表：
 - `vmstorage`：容器名稱（或主機名稱）
 - `8428`：`vmstorage` 的 HTTP 監聽端口
- `vmagent` 會定期向 `http://vmstorage:8428/metrics` 發送 HTTP 請求，並收集返回的監控數據。

2. `vmstorage` 的 `/metrics` 端點

當 `vmagent` 訪問 `http://vmstorage:8428/metrics` 時，會獲取 `vmstorage` 內部的各種監控指標，例如：

- `process_cpu_seconds_total` (CPU 使用量)
- `process_resident_memory_bytes` (記憶體使用量)
- `vm_rows_inserted_total` (寫入數據總數)
- `vm_data_size_bytes` (存儲使用量)

3. `targets` 支援的格式

除了 `"vmstorage:8428"` 這種 **單一目標** 格式，你還可以：

3.1 監控多個服務





```
static_configs:  
  - targets:  
    - "vmstorage:8428"  
    - "vminsert:8480"  
    - "vmselect:8481"  
    - "vmagent:8429"
```

這樣 `vmagent` 會同時監控 `vmstorage`、`vminsert`、`vmselect` 和 `vmagent`。

3.2 監控多個主機

如果 `vmstorage` 運行在多台機器（假設有三台），可以這樣設定：

```
static_configs:  
  - targets:  
    - "vmstorage-1.example.com:8428"  
    - "vmstorage-2.example.com:8428"  
    - "vmstorage-3.example.com:8428"
```

這樣 `vmagent` 會同時監控 **三台** `vmstorage` 節點。

3.3 使用 IP 地址

```
static_configs:  
  - targets:  
    - "192.168.1.10:8428"  
    - "192.168.1.11:8428"
```

當 `vmstorage` 沒有 DNS 設定時，可以直接使用 IP 監控。

4. `scrape_interval` 來調整抓取頻率

預設情況下，`vmagent` 每 **10-15 秒** 抓取一次 `metrics`，但你可以修改 `scrape_interval` 來改變抓取頻率，例如：



```
scrape_configs:
  - job_name: "vmstorage"
    scrape_interval: 30s # 每 30 秒抓取一次數據
    static_configs:
      - targets: ["vmstorage:8428"]
```

這樣 `vmagent` 會 **每 30 秒抓取一次 `vmstorage` 的指標**，而不是預設的 15 秒。

5. 測試 `vmstorage:8428` 是否可訪問

5.1 使用 `curl` 測試

執行以下命令來檢查 `vmstorage` 是否能正確提供 `metrics`：

```
curl -s http://vmstorage:8428/metrics | head -n 20
```

如果 `vmstorage` 運行正常，你應該會看到類似的輸出：

```
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 120 57
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 2567894528
...
```

5.2 使用 `docker logs` 檢查 `vmagent`

如果 `vmagent` 無法抓取 `vmstorage` 的數據，可以查看 `vmagent` 的日誌：

```
docker logs -f vmagent
```

如果 `vmagent` 抓取失敗，可能會顯示錯誤：



```
level=error msg="scraping target='vmstorage:8428' failed: context deadline exceeded"
```

這表示 `vmagent` 無法連接 `vmstorage`，需要檢查：

- `vmstorage` 是否正常運行
- `vmstorage` 的端口 `8428` 是否開放
- `vmagent` 是否能解析 `vmstorage` 的 DNS

6. 總結

| 配置 | 說明 |
|--------------------------------------|---|
| `static_configs:` | 定義靜態監控目標 |
| `targets: ["vmstorage:8428"]` | 監控 `vmstorage`，從 `8428` 端口抓取 `/metrics` |
| `scrape_interval: 15s` | 設定抓取頻率，每 15 秒抓取一次 |
| `scrape_configs:` | 定義 `vmagent` 的 `scrape` 任務 |
| `job_name: "vmstorage"` | 給 `vmstorage` 監控任務命名 |
| `curl http://vmstorage:8428/metrics` | 測試 `vmstorage` 是否回應 |
| `docker logs -f vmagent` | 查看 `vmagent` 是否成功抓取數據 |

這樣你就能成功 讓 `vmagent` 監控 `vmstorage`，並定期收集其運行狀況！ 🚀



以繁體中文詳細說明:有時候grafana chart讀不到vmstorage, vminsert, vmselect，只讀到vmagent





Grafana 讀取不到 `vmstorage`、`vminsert`、`vmselect` 但能讀取 `vmagent` 的可能原因及解決方法

問題描述

有時候 Grafana 的 Chart 無法讀取 `vmstorage`、`vminsert`、`vmselect` 的指標，但仍能讀取 `vmagent`，可能的原因包括：

1. `vmagent` 無法抓取 `vmstorage`、`vminsert`、`vmselect` 的 `/metrics`
2. `vmagent` 成功抓取數據但 `vminsert` 沒有存儲
3. `vmselect` 無法查詢 `vmstorage` 的數據
4. Grafana 配置錯誤，未連接到 `vmselect`
5. 容器之間的網絡問題

1. `vmagent` 是否成功抓取 `vmstorage`、`vminsert`、`vmselect`？

步驟 1：手動測試 `vmagent` 是否能抓取 `vmstorage`

```
curl -s http://vmstorage:8428/metrics | head -n 20
```

如果 `vmstorage` 正常運行，你應該會看到類似的輸出：

```
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 120 57
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 2567894528
```

如果 `curl` 命令超時或返回 `Connection refused`，請檢查：

- `vmstorage` 是否正常運行：





```
docker ps | grep vmstorage
```

- `vmstorage` 容器的端口 `8428` 是否正確開放：

```
docker inspect vmstorage | grep "8428"
```

步驟 2：檢查 `vmagent` 的日誌

```
docker logs -f vmagent
```

如果 `vmagent` 無法連接 `vmstorage`，可能會顯示錯誤：

```
level=error msg="scraping target='vmstorage:8428' failed: context deadline exceeded"
```

解決方案

如果 `vmagent` 連接失敗，請確保 `scrape_config.yml` 中有正確的 `targets` 設定：

```
scrape_configs:  
  - job_name: "vmstorage"  
    static_configs:  
      - targets: ["vmstorage:8428"]  
  
  - job_name: "vminsert"  
    static_configs:  
      - targets: ["vminsert:8480"]  
  
  - job_name: "vmselect"  
    static_configs:  
      - targets: ["vmselect:8481"]  
  
  - job_name: "vmagent"  
    static_configs:  
      - targets: ["vmagent:8429"]
```

並確保 `vmagent` 在 `docker-compose.yml` 中正確加載：





```
vmagent:  
  image: victoriametrics/vmagent  
  volumes:  
    - ./scrape_config.yml:/etc/vmagent.yml  
  command:  
    - "-promscrape.config=/etc/vmagent.yml"
```

重新啟動：

```
docker-compose down && docker-compose up -d
```

2. `vmagent` 是否成功寫入 `vminsert`？

如果 `vmagent` 可以抓取 `vmstorage`、`vminsert`、`vmselect`，但 Grafana 無法讀取數據，可能是 `vmagent` 沒有將數據成功寫入 `vminsert`。

步驟 1：檢查 `vmagent` 是否有寫入錯誤

```
docker logs -f vmagent
```

如果 `vmagent` 無法寫入 `vminsert`，可能會看到：

```
level=error msg="remote write to http://vminsert:8480/insert/ failed:  
connection refused"
```

解決方案

確保 `docker-compose.yml` 中 `vmagent` 正確寫入 `vminsert`：

```
vmagent:  
  command:  
    - "-remoteWrite.url=http://vminsert:8480/insert/"
```





然後手動測試：

```
curl -X POST "http://vminsert:8480/insert/" -d 'some_test_data'
```

如果 `vminsert` 回應錯誤，請檢查 `vminsert` 是否運行：

```
docker ps | grep vminsert
```

3. `vmselect` 是否能查詢 `vmstorage`？

如果 `vmagent` 成功將數據寫入 `vminsert`，但 Grafana 無法查詢，可能是 `vmselect` 無法從 `vmstorage` 讀取數據。

步驟 1：測試 `vmselect` 是否能查詢數據

```
curl -s "http://vmselect:8481/select/0/prometheus/api/v1/query?query=up"
```

如果 `vmselect` 連接失敗，可能會返回：

```
"error": "could not fetch data from vmstorage"
```

這表示 `vmselect` 無法連接 `vmstorage`。

解決方案

檢查 `docker-compose.yml`：

```
vmselect:  
  command: "-storageNode=vmstorage:8400"
```

並確保 `vmstorage` 正確啟動：





```
docker ps | grep vmstorage
```

如果 `vmselect` 仍然無法連接 `vmstorage`，請嘗試重啟：

```
docker restart vmstorage vmselect
```

4. Grafana 配置錯誤

如果 `vmagent` 正確抓取數據，且 `vmselect` 能夠查詢數據，但 **Grafana 還是讀不到 `vmstorage`、`vminsert`、`vmselect`**，請檢查 Grafana 是否連接到 `vmselect`。

步驟 1：檢查 Grafana 的 `Data Sources`

1. 進入 Grafana → `Configuration` → `Data Sources`
2. 點擊 `Prometheus`
3. 檢查 `URL`：

```
http://vmselect:8481/select/0/prometheus
```

4. 點擊 `Save & Test`，如果連接失敗：

- 檢查 `vmselect` 是否在運行：

```
docker ps | grep vmselect
```

- 檢查 `vmselect` 是否可訪問：

```
curl -s http://vmselect:8481/select/0/prometheus/api/v1/query?  
query=up
```

- 如果 `vmselect` 連接正常，則 Grafana 配置應該沒有問題。

5. 檢查容器網絡





如果所有服務都在 **Docker Compose** 的 `network` 中，但仍然無法互相連接，可能是 容器之間的網絡問題。

步驟 1：檢查 Docker 網絡

```
docker network inspect monitoring
```

如果 `vmstorage`、`vminsert`、`vmselect` 不在 `monitoring` 網絡，請更新 `docker-compose.yml`：

```
networks:  
  monitoring:
```

並確保每個服務都連接到 `monitoring`：

```
vmselect:  
  networks:  
    - monitoring
```

然後重啟：

```
docker-compose down && docker-compose up -d
```

總結

| 問題 | 解決方案 |
|--------------------------------|--|
| `vmagent` 無法抓取 `vmstorage` | 檢查 `scrape_config.yml`，確認 `vmstorage:8428` 可訪問 |
| `vmagent` 無法寫入 `vminsert` | 確保 `-remoteWrite.url=http://vminsert:8480/insert/` |
| `vmselect` 無法查詢 `vmstorage` | 檢查 `-storageNode=vmstorage:8400` |





| 問題 | 解決方案 |
|-------------------------|--|
| Grafana 無法讀取 `vmselect` | 確保 `Data Source URL` 為 `http://vmselect:8481/select/0/prometheus` |
| 容器網絡問題 | `docker network inspect monitoring`，確保所有容器連接到 `monitoring` |

透過這些步驟，你應該可以成功解決 **Grafana 讀不到 `vmstorage`、`vminsert`、`vmselect` 的問題** 🚀



以繁體中文詳細說明：有時候執行 docker-compose up -d 後，
vminsert 連線不到 vmstorage，重新執行 docker-compose down，
docker-compose up -d 就可以



問題：Docker Compose 啟動後，`vminsert` 連線不到 `vmstorage`，但重新 `docker-compose down && docker-compose up -d` 就正常

問題描述

當執行 `docker-compose up -d` 後，`vminsert` 有時候無法連線到 `vmstorage`，但重新執行 `docker-compose down && docker-compose up -d` 後就恢復正常。這種情況可能是由於 **服務啟動順序、Docker 網絡問題、DNS 解析延遲** 或 `vmstorage` 未準備好 所導致。

1. 可能的原因

原因 1：`vminsert` 啟動太快，`vmstorage` 尚未準備好

`vmstorage` 負責存儲數據，當 `docker-compose` 啟動所有容器時：

1. `vminsert` 可能比 `vmstorage` 啟動得更快，導致 `vminsert` 嘗試連線 `vmstorage` 時，`vmstorage` 仍未啟動完成。





2. `vminsert` 會因為 `vmstorage` 未準備好而無法連接。

解決方案：

- 在 `vminsert` 配置 `depends_on` 確保它在 `vmstorage` 啟動後才啟動。
 - 使用 `restart: always` 讓 `vminsert` 在失敗後重啟，避免啟動順序影響。
-

原因 2：Docker Compose 預設 `depends_on` 不等待應用完成啟動

Docker Compose 的 `depends_on` 僅確保 `vmstorage` 容器被啟動，但不確保 `vmstorage` 內部應用已準備好。這意味著：

- `vmstorage` 可能已經 **啟動容器**，但仍在初始化，未能立即提供服務。
- `vminsert` 嘗試連線時，可能 `vmstorage` 仍未準備好，導致連線失敗。

解決方案：

- 使用 `healthcheck` 確保 `vmstorage` **真正準備好** 後再啟動 `vminsert`。
-

原因 3：Docker 網絡或 DNS 延遲

當 `docker-compose up -d` 啟動容器時：

- `vminsert` 需要透過 Docker 的內部 DNS **解析** `vmstorage`。
- 有時候 DNS 解析延遲**，導致 `vminsert` 嘗試連接 `vmstorage` 但找不到 IP。

解決方案：

- 讓 `vminsert` **等待** `vmstorage` **準備好後再連線**，避免因為 DNS 解析延遲而導致的連線錯誤。
-

2. 解決方案



解決方案 1：在 `vminsert` 設定 `depends_on`

在 `docker-compose.yml` 中，修改 `vminsert`，確保它在 `vmstorage` 之後啟動：

```
version: '3.7'
services:
  vmstorage:
    image: victoriametrics/vmstorage
    container_name: vmstorage
    restart: always
    ports:
      - "8428:8428"
    networks:
      - monitoring

  vminsert:
    image: victoriametrics/vminsert
    container_name: vminsert
    restart: always
    depends_on:
      - vmstorage
    command: -storageNode=vmstorage:8400
    ports:
      - "8480:8480"
    networks:
      - monitoring
```

“限制：`depends_on` 不能確保 `vmstorage` 應用層已經準備好，它只確保 `vmstorage` 容器已經啟動。”

解決方案 2：在 `vmstorage` 設定 `healthcheck`

方式 1：使用 `healthcheck` 確保 `vmstorage` 準備好

```
vmstorage:
  image: victoriametrics/vmstorage
  container_name: vmstorage
  restart: always
```





```
healthcheck:  
  test: ["CMD", "curl", "-f", "http://localhost:8428/metrics"]  
  interval: 5s  
  retries: 5  
  start_period: 10s  
ports:  
  - "8428:8428"  
networks:  
  - monitoring
```

方式 2：讓 `vminsert` 等待 `vmstorage` 準備好

```
vminsert:  
  image: victoriametrics/vminsert  
  container_name: vminsert  
  restart: always  
  depends_on:  
    - vmstorage:  
        condition: service_healthy  
  command: -storageNode=vmstorage:8400  
  ports:  
    - "8480:8480"  
  networks:  
    - monitoring
```

這樣 `vminsert` 只會在 `vmstorage`、`healthcheck` 成功後才會啟動。

解決方案 3：在 `vminsert` 添加啟動等待機制

如果 `healthcheck` 無法使用，你可以讓 `vminsert` 在啟動時等待 10 秒再連線：

```
vminsert:  
  image: victoriametrics/vminsert  
  container_name: vminsert  
  restart: always  
  depends_on:  
    - vmstorage  
  entrypoint: ["/bin/sh", "-c", "sleep 10; exec /victoria-metrics-prod -"]
```





```
storageNode=vmstorage:8400"]  
ports:  
- "8480:8480"  
networks:  
- monitoring
```

這會讓 `vminsert` 在啟動後等待 **10** 秒，確保 `vmstorage` 已完全準備好。

解決方案 4：使用 `restart: always`

如果 `vminsert` 因為 `vmstorage` 尚未準備好而崩潰，Docker 會嘗試重新啟動它。因此，你可以確保 `restart: always` 設定正確：

```
vminsert:  
image: victoriametrics/vminsert  
container_name: vminsert  
restart: always  
depends_on:  
- vmstorage  
command: -storageNode=vmstorage:8400  
ports:  
- "8480:8480"  
networks:  
- monitoring
```

這樣 `vminsert` 會在 `vmstorage` 準備好後 **自動重新啟動**，避免連線失敗。

3. 總結

問題

解決方案

| | |
|-----------------------------------|------------------------------------|
| `vminsert` 啟動太快，`vmstorage` 尚未準備好 | 使用 `depends_on` 確保 `vmstorage` 先啟動 |
|-----------------------------------|------------------------------------|





| 問題 | 解決方案 |
|---|--|
| `depends_on` 無法確保 `vmstorage` 應用層準備好 | 使用 `healthcheck` 確保 `vmstorage` 真的準備好 |
| `vmagent` 寫入 `vminsert` 失敗，因為 `vminsert` 啟動太快 | 讓 `vminsert` 在啟動後等待 10 秒 |
| `vminsert` 連線失敗，Docker 網絡問題 | 檢查 `docker network inspect monitoring` 確保所有容器在同一網絡 |
| `vminsert` 啓動後崩潰 | 設定 `restart: always` 讓 `vminsert` 自動重啟 |

最推薦的方法：

1. 設定 `depends_on: condition: service_healthy`
2. 在 `vmstorage` 設定 `healthcheck`
3. 讓 `vminsert` `entrypoint` 延遲啟動 10 秒
4. 使用 `restart: always` 確保 `vminsert` 能夠重啟

這樣你可以避免 `vminsert` 在 `vmstorage` 尚未準備好時嘗試連線，從而解決 必須手動 `docker-compose down && up` 的問題！



以繁體中文詳細說明: [vmui用法](#)



VictoriaMetrics UI (`vmui`) 使用詳解

`vmui` 是 VictoriaMetrics 提供的內建 Web UI，用於查詢、監控和管理時序數據。

這個 UI 與 Prometheus 的 Web UI 類似，允許用戶：

- **查詢時序數據 (PromQL)**
- **檢視儲存的 metrics**
- **檢查 `vmstorage` 狀態**
- **監控 VictoriaMetrics 內部指標**





1. 如何訪問 `vmui`

1.1 `vmui` 預設端口

VictoriaMetrics 的 Web UI (`vmui`) 預設監聽 **8428** 端口（單節點）或 **8481**（`vmselect`）。

1.2 訪問 `vmui`

如果你在本機執行 VictoriaMetrics（單節點模式）：

```
http://localhost:8428/vmui
```

如果你使用 **VictoriaMetrics 集群模式**，則應該連接 `vmselect`：

```
http://vmselect:8481/vmui
```

如果你在 **Docker Compose 部署**，可以找到 `vmui`：

```
docker inspect vmstorage | grep "IPAddress"
```

然後打開：

```
http://<vmstorage-ip>:8428/vmui
```

2. `vmui` 主要功能

`vmui` 提供了一個簡單的 Web 介面，包含以下主要功能：

| 功能 | 說明 |
|---------------|---|
| 查詢數據 (PromQL) | 使用 Prometheus Query Language (PromQL) 來查詢數據 |





| 功能 | 說明 |
|-------------------------|------------------------------------|
| 查看可用的 metrics | 列出所有 VictoriaMetrics 存儲的 metric 名稱 |
| 檢視 VictoriaMetrics 內部狀態 | 查看當前系統資源使用狀況 |
| 儲存查詢結果 | 可以將查詢結果另存為 JSON |
| 監控 TSDB (時序數據庫) 狀態 | 包含 `vmstorage`、`vmselect` 內部指標 |
| 遠端讀取 (Remote Read) | 查詢並讀取遠端 Prometheus-style TSDB 數據 |

3. `vmui` 查詢數據 (PromQL)

VictoriaMetrics 完全兼容 **Prometheus Query Language (PromQL)**，所以你可以使用 `vmui` 查詢時序數據。

3.1 査詢某個 Metric

在 `vmui` 查詢框內輸入：

```
up
```

這將顯示 **所有存活 (Up)** 的目標。

3.2 過去 5 分鐘內的 CPU 使用率

```
rate(process_cpu_seconds_total[5m])
```

這將返回 **最近 5 分鐘內 `process_cpu_seconds_total` 的變化率**。

3.3 監控 `vmstorage` 寫入數據的速率

```
rate(vm_rows_inserted_total[5m])
```

- `vm_rows_inserted_total`：總共寫入的數據行數





- ``rate(...[5m])``: 計算 5 分鐘內的變化率 (每秒寫入多少行數據)

3.4 監控 `vmselect` 查詢延遲

```
rate(vm_select_duration_seconds_sum[5m]) /  
rate(vm_select_duration_seconds_count[5m])
```

- `vm_select_duration_seconds_sum`：累積的查詢時間總和
 - `vm_select_duration_seconds_count`：查詢次數
 - **結果**：每次查詢的平均時間（秒）

4. `vmui` 查看已存儲的 metrics

在 `vmut`，點擊 「Browse active time series」，可以查看所有存儲的 metrics。

你也可以手動查詢：

```
{__name__=~".*"}}
```

這將列出 **所有 metrics**, 類似於 Prometheus `http://localhost:9090/api/v1/series?match[1]=.*`。

5. 監控 VictoriaMetrics 內部狀態

VictoriaMetrics 本身會提供 **內部監控指標**，你可以在 `vmui` 查詢這些指標來監控系統狀態。

5.1 查看 VictoriaMetrics 內部使用的記憶體

process_resident_memory_bytes



這會顯示 當前使用的記憶體大小 (bytes)。

5.2 檢查 VictoriaMetrics 當前存儲的數據大小

```
vm_data_size_bytes
```

這會顯示 當前佔用的磁碟空間 (bytes)。

5.3 監測 `vmagent` 寫入數據的速率

```
rate(vm_insert_duration_seconds_sum[5m]) /  
rate(vm_insert_duration_seconds_count[5m])
```

- 這可以監測 `vmagent` 每筆數據的寫入時間。

6. 如何使用 `vmui` 監控 VictoriaMetrics

你可以 將 `vmui` 數據整合到 Grafana，用來監控 `vmstorage`、`vminsert`、`vmselect` 的健康狀態。

6.1 在 Grafana 設置 `vmselect` 作為數據源

1. 打開 Grafana
2. 添加數據源 → `Prometheus`
3. 設置 URL

```
http://vmselect:8481/select/0/prometheus
```

4. 點擊「Save & Test」確保 Grafana 能夠連線 `vmui`。

6.2 在 Grafana 添加監控面板

1. 打開 Dashboard
2. 添加 Panel





3. 輸入 PromQL 查詢

- CPU 使用率：

```
rate(process_cpu_seconds_total[5m])
```

- 記憶體使用：

```
process_resident_memory_bytes
```

- 磁碟使用：

```
vm_data_size_bytes
```

7. API 查詢方式（適用於自動化監控）

如果你希望通過 API 獲取 `vmui` 查詢結果，可以使用以下方式：

7.1 查詢當前存活的 targets

```
curl "http://vmstorage:8428/api/v1/query?query=up"
```

7.2 查詢當前存儲的 metric

```
curl "http://vmstorage:8428/api/v1/series?match[]={__name__=~'.*'}"
```

7.3 查詢過去 1 小時內的 `vmstorage` CPU 使用

```
curl "http://vmstorage:8428/api/v1/query?  
query=rate(process_cpu_seconds_total[1h])"
```

8. `vmui` 可能遇到的錯誤與解決辦法





| 錯誤 | 可能原因 | 解決辦法 |
|-------------|------------------|--|
| `vmui` 無法訪問 | `vmstorage` 未啟動 | `docker ps` |
| `vmui` 沒有數據 | `vmagent` 沒有抓取數據 | 檢查 `scrape_config.yml` 確保 `vmagent` 正確抓取 |
| 查詢超時 | `vmselect` 負載過高 | 增加 `vmselect` CPU/記憶體，或減少查詢範圍 |

9. 總結

| 功能 | 指令 |
|----------------|--|
| 訪問 `vmui` | `http://localhost:8428/vmui` |
| 查詢所有 `metrics` | `\$__name__=~".*"` |
| 查詢 CPU 使用率 | `rate(process_cpu_seconds_total[5m])` |
| 查詢記憶體使用 | `process_resident_memory_bytes` |
| 查詢磁碟使用 | `vm_data_size_bytes` |
| API 查詢 | `curl "http://vmstorage:8428/api/v1/query?query=up"` |

透過 `vmui`，你可以快速 **查詢 VictoriaMetrics 數據、監控系統狀態，並與 Grafana 整合進行可視化分析！** 🚀



以繁體中文詳細說明:PromQL的rate與其他常用func



PromQL (`Prometheus Query Language`) 常用函數詳解

PromQL 是 Prometheus 及其兼容數據庫（如 **VictoriaMetrics**）使用的查詢語言，用於分析時序數據（time-series data）。





本文將詳細介紹 `rate()` 以及其他常用的 PromQL 函數，幫助你在 **VictoriaMetrics** (`vmui`)、**Grafana** 等工具中查詢數據。

1. `rate()` - 計算速率 (增量/秒)

`rate()` 用於計算 **Counter** 指標的變化率 (增量/秒)，適用於 **單調遞增的指標** (如總請求數、寫入數量等)。

語法

```
rate(metric_name[時間範圍])
```

範例

1.1 計算 5 分鐘內 `http_requests_total` 的請求速率

```
rate(http_requests_total[5m])
```

解釋：

- `http_requests_total` 是一個 **Counter** (單調遞增的指標)。
- `rate(...[5m])` 計算 **過去 5 分鐘的請求速率** (每秒請求數)。

1.2 計算 `vmstorage` 每秒寫入行數

```
rate(vm_rows_inserted_total[5m])
```

- `vm_rows_inserted_total` 是 **VictoriaMetrics** 寫入的總數據行數 (單調遞增)。
- `rate(...[5m])` 計算 **最近 5 分鐘內每秒寫入多少行數據**。

注意事項

- `rate()` 僅適用於 **Counter** (單調遞增的指標)。
- `rate()` 計算的是 **每秒變化量**，單位通常是 **count/second**。





2. `increase()` - 計算變化總量

`increase()` 計算 Counter 在指定時間範圍內的變化總量。

語法

```
increase(metric_name[時間範圍])
```

範例

2.1 計算 `http_requests_total` 在 1 小時內的總增量

```
increase(http_requests_total[1h])
```

- `increase(...[1h])` 計算 過去 1 小時內的請求總數。

2.2 計算 `vmstorage` 在一天內寫入的總行數

```
increase(vm_rows_inserted_total[1d])
```

- `increase(...[1d])` 計算 過去 24 小時內 VictoriaMetrics 存儲的總行數。

`increase()` vs `rate()`

| 函數 | 功能 | 單位 |
|------------------------|--------------|--------------|
| `rate(metric[5m])` | 計算每秒變化量 | count/second |
| `increase(metric[5m])` | 計算這段時間內的變化總量 | count |

3. `avg_over_time()` - 計算區間內的平均值

`avg_over_time()` 計算某個 Gauge 指標在指定時間範圍內的平均值。





語法

```
avg_over_time(metric_name[時間範圍])
```

範例

3.1 計算 `CPU` 在 10 分鐘內的平均使用率

```
avg_over_time(process_cpu_seconds_total[10m])
```

- `process_cpu_seconds_total` 是 **CPU** 使用時間累計值。
- `avg_over_time(...[10m])` 計算 **最近 10 分鐘的 CPU 平均使用量**。

3.2 計算 `vmstorage` 在 1 小時內的平均存儲使用量

```
avg_over_time(vm_data_size_bytes[1h])
```

- `vm_data_size_bytes` 是 **當前存儲大小** (Gauge)。
- `avg_over_time(...[1h])` 計算 **最近 1 小時內的平均存儲大小**。

4. `max_over_time()` & `min_over_time()` - 計算區間內的最大/最小值

這兩個函數用於找出某個數據在指定時間範圍內的最大值或最小值。

語法

```
max_over_time(metric_name[時間範圍])
min_over_time(metric_name[時間範圍])
```

範例

4.1 查詢 `CPU` 在 10 分鐘內的最高使用率





```
max_over_time(process_cpu_seconds_total[10m])
```

4.2 查詢 `vmstorage` 在 1 小時內的最低存儲大小

```
min_over_time(vm_data_size_bytes[1h])
```

5. `quantile_over_time()` - 計算區間內的分位數

`quantile_over_time()` 計算某個指標在指定時間範圍內的分位數 (Percentile)。

語法

```
quantile_over_time(分位數, metric_name[時間範圍])
```

- `0.5` (50%) : 中位數
- `0.9` (90%) : 90% 分位數
- `0.99` (99%) : 99% 分位數

範例

5.1 查詢 `vmselect` 查詢時間的 P99 (99% 分位數)

```
quantile_over_time(0.99, vm_select_duration_seconds[5m])
```

- P99 表示 99% 的查詢時間低於這個數值。

6. `deriv()` - 計算瞬時變化率

`deriv()` 計算某個數據的導數 (瞬時變化率)。





語法

```
deriv(metric_name[時間範圍])
```

範例

6.1 計算 `vmstorage` 存儲大小的瞬時變化率

```
deriv(vm_data_size_bytes[10m])
```

- `deriv(...[10m])` 計算 過去 10 分鐘的存儲大小變化率。

7. `predict_linear()` - 預測未來數據

`predict_linear()` 可用於預測數據的未來趨勢。

語法

```
predict_linear(metric_name[時間範圍], 預測秒數)
```

範例

7.1 預測 `vmstorage` 存儲大小未來 1 小時的變化

```
predict_linear(vm_data_size_bytes[1h], 3600)
```

- 這將預測 1 小時後 `vmstorage` 佔用的磁碟大小。

8. 總結





| 函數 | 功能 | 適用類型 |
|--|------------|----------------------|
| `rate(metric[5m])` | 計算速率（增量/秒） | Counter |
| `increase(metric[5m])` | 計算總變化量 | Counter |
| `avg_over_time(metric[5m])` | 計算區間內的平均值 | Gauge |
| `max_over_time(metric[5m])` | 計算區間內的最大值 | Gauge |
| `min_over_time(metric[5m])` | 計算區間內的最小值 | Gauge |
| `quantile_over_time(0.99, metric[5m])` | 計算區間內 P99 | Gauge |
| `deriv(metric[5m])` | 計算瞬時變化率 | Counter/Gauge |
| `predict_linear(metric[5m], 3600)` | 預測未來趨勢 | Gauge |

透過這些 **PromQL** 函數，你可以 分析 **VictoriaMetrics / Prometheus** 中的時序數據，並在 `vmui` 或 **Grafana** 中監控系統狀態！🚀



以繁體中文詳細說明:metric



Prometheus / VictoriaMetrics 的 Metric 詳細說明

在 **Prometheus** 或 **VictoriaMetrics** 中，`metric`（指標）是 監控數據的基本單位，它代表 某個測量值隨時間變化的記錄，例如：

- CPU 使用率
- 記憶體佔用量
- 每秒請求數
- 數據存儲量

這些 `metric` 會被持續收集，並以 **時序數據 (Time Series Data)** 的形式存儲。





1. `metric` 的結構

一個完整的 `metric` 由以下 **三個部分** 組成：

1. **Metric 名稱 (Name)**
2. **標籤 (Labels)**
3. **時間戳 (Timestamp) 和數值 (Value)**

1.1 `metric` 的格式

```
metric_name{label1="value1", label2="value2"} value timestamp
```

“示例：”

```
http_requests_total{method="GET", status="200"} 1000 1677859200
```

這條 `metric` 代表：

- **名稱**：`http_requests_total`（總 HTTP 請求數）
- **標籤**：
 - `method="GET"`（HTTP 方法）
 - `status="200"`（HTTP 狀態碼）
- **數值**：1000（累積的請求數）
- **時間戳**：1677859200（記錄這條數據的時間點）

2. `metric` 的四種類型

在 Prometheus 和 VictoriaMetrics 中，`metric` 分為 **四種類型**：

| 類型 | 作用 | 使用範例 |
|----------------------|----------|----------------------------------|
| Counter (計數器) | 單調遞增的計數值 | `http_requests_total`（HTTP 請求總數） |





| 類型 | 作用 | 使用範例 |
|-----------------|-------------------|--|
| Gauge (儀表) | 可上升或下降的測量值 | `memory_usage_bytes` (記憶體使用量) |
| Histogram (直方圖) | 記錄一組數值的分佈 | `request_duration_seconds_bucket` (請求延遲分佈) |
| Summary (摘要統計) | 記錄統計摘要 (如 P99 延遲) | `http_request_duration_seconds` (請求時間摘要) |

3. `metric` 類型詳細說明

3.1 Counter (計數器)

- Counter 只會增加，不會減少 (除非重啟)。
- 典型用途：
 - 計算 總請求數
 - 計算 總錯誤數
 - 計算 總寫入數據行數

示例：

```
http_requests_total{method="GET"} 12345
```

常見 Counter 指標

| Metric | 作用 |
|-----------------------------|---------------------------|
| `http_requests_total` | HTTP 請求總數 |
| `vm_rows_inserted_total` | 寫入到 VictoriaMetrics 的數據行數 |
| `process_cpu_seconds_total` | 進程消耗的 CPU 總時間 |

如何查詢 Counter 的變化率

```
rate(http_requests_total[5m])
```





這個查詢會 計算最近 5 分鐘內的請求速率（每秒請求數）。

3.2 Gauge (儀表)

- **Gauge** 代表可增可減的數值（不像 Counter 只增不減）。
- 典型用途：
 - 記錄 **記憶體使用量**
 - 記錄 **當前 CPU 使用率**
 - 記錄 **當前磁碟使用量**

示例：

```
memory_usage_bytes 20480000000
```

常見 Gauge 指標

| Metric | 作用 |
|---------------------------------|------------------------|
| `process_resident_memory_bytes` | 進程的記憶體佔用量 |
| `vm_data_size_bytes` | VictoriaMetrics 存儲的數據量 |
| `node_cpu_seconds_total` | 當前 CPU 佔用率 |

如何查詢 Gauge 在過去 10 分鐘的平均值

```
avg_over_time(process_resident_memory_bytes[10m])
```

3.3 Histogram (直方圖)

- **Histogram** 會統計數據的分佈情況（例如請求時間的分佈）。
- 包含三種指標：





1. `_bucket` : 不同區間的數據量
2. `_sum` : 總數據量
3. `_count` : 數據點數量

示例

```
http_request_duration_seconds_bucket{le="0.1"} 1024
http_request_duration_seconds_bucket{le="0.5"} 5000
http_request_duration_seconds_bucket{le="1"} 8000
http_request_duration_seconds_bucket{le="+Inf"} 10000
```

這代表：

- **1024 個請求** 的執行時間 小於等於 **0.1 秒**
- **5000 個請求** 的執行時間 小於等於 **0.5 秒**
- **8000 個請求** 的執行時間 小於等於 **1 秒**
- **10000 個請求** 是所有請求總數

常見 Histogram 指標

| Metric | 作用 |
|--|---------------------------|
| `http_request_duration_seconds_bucket` | HTTP 請求時間分佈 |
| `vm_insert_duration_seconds_bucket` | VictoriaMetrics 插入數據的時間分佈 |

如何查詢 P95 (95% 分位數)

```
histogram_quantile(0.95, rate(http_request_duration_seconds_bucket[5m]))
```

3.4 Summary (摘要統計)

- **Summary** 提供統計數據 (如 **P99, P95**)，但不像 Histogram 具有 `bucket`，而是直接提供 `_sum` 和 `_count`。
- 包含三種指標：





1. `_sum` : 數據總和
2. `_count` : 數據數量
3. `_quantile` : 分位數 (如 `0.5` , `0.95` , `0.99`)

示例

```
http_request_duration_seconds{quantile="0.5"} 0.2
http_request_duration_seconds{quantile="0.95"} 1.5
http_request_duration_seconds{quantile="0.99"} 2.8
```

這代表：

- **50% (P50)** 的請求時間小於 **0.2** 秒
- **95% (P95)** 的請求時間小於 **1.5** 秒
- **99% (P99)** 的請求時間小於 **2.8** 秒

常見 Summary 指標

| Metric | 作用 |
|---------------------------------|------------------------|
| `http_request_duration_seconds` | HTTP 請求時間摘要 |
| `vm_select_duration_seconds` | VictoriaMetrics 查詢時間摘要 |

如何查詢平均請求時間

```
rate(http_request_duration_seconds_sum[5m]) /
rate(http_request_duration_seconds_count[5m])
```

4. 如何查詢所有 `metric`

在 **VictoriaMetrics UI** (`vmui`) 或 **Prometheus Web UI** 中，使用以下查詢來列出所有 `metric`：



```
{__name__=~".*"}  
 
```

在 API 查詢方式

```
curl "http://vmstorage:8428/api/v1/series?match[]={__name__=~'.*'}"  
 
```

5. 總結

| 類型 | 作用 | 例子 |
|-----------|------|--|
| Counter | 單調遞增 | `http_requests_total` |
| Gauge | 可增可減 | `process_resident_memory_bytes` |
| Histogram | 數據分佈 | `http_request_duration_seconds_bucket` |
| Summary | 統計數據 | `http_request_duration_seconds` |

這些 `metric` 是 Prometheus 和 VictoriaMetrics 核心的監控數據，透過 `PromQL` 查詢，你可以 分析系統性能，監控資源使用，並進行告警設定！🚀



以繁體中文詳細說明:grafana複製panels



Grafana 複製 Panels 的詳細說明

在 Grafana 中，Panel（面板）是儀表板（Dashboard）的基本組件，用來顯示監控數據的圖表、表格或指標視覺化。當你需要 複製 Panel，可以使用以下幾種方法：

1. 在同一個 Dashboard 內複製 Panel
2. 將 Panel 複製到不同的 Dashboard
3. 將 Panel 設定另存為 JSON，手動匯入





4. 批量複製多個 Panels

方法 1：在同一個 Dashboard 內複製 Panel

如果你想 在當前 Dashboard 複製一個 Panel，請按照以下步驟操作：

步驟 1：打開要複製的 Panel

1. 打開 Grafana，進入你的 Dashboard。
2. 移動滑鼠到要複製的 Panel 上。
3. 點擊 Panel 標題右上角的「:」圖示（更多選項）。

步驟 2：選擇「Duplicate」

1. 在選單中，點擊「Duplicate」（複製）。
2. Grafana 會立即在 相同 Dashboard 中創建一個 完全相同的 Panel。

步驟 3：修改 Panel（如果需要）

1. 你可以點擊 Panel 進入編輯模式（`Edit`），修改查詢、時間範圍或 Panel 樣式。
2. 點擊 `Apply`（應用）儲存變更。

這個方法適用於：

- 在 相同的 Dashboard 內 快速複製 Panel。
- 只需對部分 Panel 進行調整（如查詢 PromQL 不同指標）。

方法 2：將 Panel 複製到不同的 Dashboard

如果你想 將一個 Panel 複製到另一個 Dashboard，請使用「複製並移動」功能。

步驟 1：打開 Panel 設定





1. 在 **Dashboard** 中，移動滑鼠到 Panel 上。
2. 點擊 「:」 (更多選項)。
3. 選擇 「**Copy**」 (複製)。

步驟 2：前往目標 Dashboard

1. 進入 **目標 Dashboard**。
2. 點擊 「**Add panel**」 (新增 Panel)。
3. 選擇 「**Paste**」 (貼上)。

步驟 3：調整 Panel 設定

1. Panel 會被複製到新的 Dashboard。
2. 你可以點擊 **Edit** (編輯) 來調整查詢、時間範圍或視覺化方式。

這個方法適用於：

- 在 **不同的 Dashboard** 之間 複製 Panel。
- **保持相同的查詢與設定**，但應用到不同的監控場景。

方法 3：使用 JSON 決出 & 匯入

如果你需要 **手動導入 Panel**，或者在不同的 Grafana 伺服器之間共享 Panel，可以使用 **JSON 決出**。

步驟 1：匯出 Panel JSON

1. 點擊 **Panel** 「:」 → **Edit** (編輯)。
2. 在右上角選擇 「**JSON Model**」。
3. 複製 **Panel JSON** 配置。

步驟 2：在目標 Dashboard 匯入 JSON

1. 進入 **目標 Dashboard**。





2. 點擊「:」→ **Edit** (編輯)。
3. 切換到「**JSON Model**」，貼上 JSON 配置。
4. 點擊「**Apply**」(應用)。

這個方法適用於：

- 跨伺服器複製 **Panel** (不同的 Grafana 環境)。
 - 手動調整 **Panel** 設定，例如修改 Data Source。
-

方法 4：批量複製多個 Panels

如果你想一次複製整個 **Dashboard** 或多個 **Panels**，可以使用「匯出/匯入 Dashboard」功能。

步驟 1：匯出 Dashboard

1. 進入 Dashboard，點擊「設定」(⚙️) → 「**JSON Model**」。
2. 點擊「**Export**」(匯出)。
3. 選擇「**With current settings**」(包含當前設定)，下載 JSON 檔案。

步驟 2：匯入到新 Dashboard

1. 進入 **Grafana** 首頁。
2. 點擊「**Create**」→「**Import**」(匯入)。
3. 上傳剛剛下載的 JSON 檔案。
4. 選擇 Data Source (數據來源)。
5. 點擊「**Import**」(匯入)。

這個方法適用於：

- 一次複製整個 **Dashboard**。
 - 跨不同 **Grafana** 伺服器共享設定。
-





5. 總結

| 方法 | 適用場景 | 操作步驟 |
|----------------------|-----------------------------|--|
| 方法 1：複製到相同 Dashboard | 在同一個 Dashboard 內 快速複製 Panel | 點擊 `⋮` → `Duplicate` |
| 方法 2：複製到不同 Dashboard | 將 Panel 複製到另一個 Dashboard | 點擊 `⋮` → `Copy` → 進入新 Dashboard `Paste` |
| 方法 3：JSON 匯出 & 匯入 | 跨伺服器共享 Panel 設定 | `⋮` → `Edit` → `JSON Model` → `Copy & Paste` |
| 方法 4：批量複製 | 一次複製多個 Panels 或整個 Dashboard | 設定 `⋮` → `Export JSON` → `Import` |

這些方法可以幫助你 **快速複製 Panel**，無論是在相同 Dashboard，還是不同的 Grafana 伺服器之間！🚀



以繁體中文詳細說明:grafana的panels



Grafana Panels 詳細說明

在 **Grafana** 中，**Panel**（面板）是 **儀表板（Dashboard）** 的核心組件，負責將監控數據轉換為視覺化圖表，如 **折線圖、長條圖、表格、單一數值（Stat）** 等。

每個 Panel 都可以設定：

1. 數據來源（Data Source）
2. 查詢方式（PromQL、SQL、InfluxDB Query 等）
3. 視覺化方式（Graph、Gauge、Table 等）
4. 時間範圍（自動更新或手動選擇）
5. 聲報設定（Alert）





1. 如何新增 Panel

方法 1：從 Dashboard 新增

1. 進入 **Grafana Dashboard**。
2. 點擊 「+」 → 「Add Panel」 (新增面板)。
3. 選擇數據來源 (例如 Prometheus、VictoriaMetrics、InfluxDB)。
4. 輸入查詢語法 (**PromQL、SQL** 等)。
5. 選擇視覺化方式 (**Graph、Stat、Table** 等)。
6. 點擊 「Apply」 (應用)， Panel 會顯示在 Dashboard 中。

方法 2：複製現有 Panel

1. 在現有 **Panel** 右上角點擊 `⋮` (更多選項)。
2. 選擇 「**Duplicate**」 (複製)，在相同 Dashboard 內新增相同的 Panel。
3. 如果要複製到另一個 Dashboard：
 - 點擊 「**Copy**」 (複製)。
 - 到新 Dashboard 「**Add Panel**」 → 「**Paste**」 (貼上)。

2. Panel 的主要組成

每個 Panel 由以下幾個部分組成：

| 組件 | 功能 |
|------------------------------|--|
| Data Source (數據來源) | 指定從哪裡獲取數據，如 Prometheus、VictoriaMetrics、MySQL 等 |
| Query (查詢語法) | 使用 PromQL、SQL、InfluxDB Query 語法等來獲取數據 |
| Visualization (視覺化類型) | 設定如何顯示數據，如折線圖、長條圖、表格、單值等 |
| Time Range (時間範圍) | 設定 Panel 的時間範圍 (相對時間或絕對時間) |
| Panel Options (選項) | 設定標題、圖表樣式、單位等 |
| Alert (警報) | 設定條件，在數據異常時發送通知 |





3. Panel 的主要視覺化類型

Grafana 提供多種 **視覺化類型 (Visualization)**，以下是最常見的 7 種 **Panel 類型**：

| 視覺化類型 | 適用場景 | 示例 |
|------------------------|----------|-------------------|
| Graph (折線圖) | 顯示時序數據變化 | 監控 CPU、記憶體、流量趨勢 |
| Stat (單值) | 顯示單一數值 | 當前 CPU 使用率、活躍用戶數 |
| Table (表格) | 顯示結構化數據 | 查詢 API 請求記錄、數據庫狀態 |
| Gauge (儀表盤) | 顯示數值範圍 | 監測 CPU 負載、磁碟使用率 |
| Bar Gauge (條狀圖) | 顯示多個指標 | 監控多台伺服器的 CPU |
| Heatmap (熱圖) | 顯示數據密度 | 監測請求延遲分佈 |
| Pie Chart (圓餅圖) | 顯示比例 | 監控錯誤類型、HTTP 狀態碼 |

3.1 Graph (折線圖)

適用於顯示隨時間變化的數據，如：

```
rate(process_cpu_seconds_total[5m])
```

示例：

- 監控 **CPU 使用率**
- 監控 **API 請求數**
- 監控 **流量趨勢**

3.2 Stat (單值)

顯示單一數值，如當前狀態：

```
process_resident_memory_bytes
```

示例：



- 監測 **目前記憶體使用量**
- 監測 **伺服器溫度**
- 監測 **線上用戶數**

3.3 Table (表格)

適用於顯示結構化數據，如：

```
topk(5, node_cpu_seconds_total)
```

示例：

- 監控 **前 5 名 CPU 使用率最高的伺服器**
- 監控 **HTTP 錯誤日誌**

3.4 Gauge (儀表盤)

適用於 **顯示某個數值在範圍內的變化**，如：

```
node_memory_Active_bytes / node_memory_MemTotal_bytes * 100
```

示例：

- 監測 **CPU 負載**
- 監測 **磁碟使用率**

4. Panel 設定 (Options & Styles)

Grafana 允許你調整 Panel 的外觀和功能，例如：

| 設定項目 | 功能 |
|--------------------|--------------|
| Title | 設定 Panel 的標題 |
| Description | 添加 Panel 的說明 |





| 設定項目 | 功能 |
|-------------------------|---------------------------|
| Unit (單位) | 設定數據單位，如 `bytes`、`ms`、`%` |
| Legend (圖例) | 設定圖例是否顯示 |
| Thresholds (警戒線) | 設定警戒值，當數據超過範圍時變色 |

5. Panel 設定警報 (Alert)

Grafana 允許你設定警報，當數據異常時發送通知（如 Email、Slack、Discord）。

步驟 1：新增 Alert

1. 打開 Panel，點擊 「Edit」（編輯）。
2. 切換到 「Alert」（警報）選單。
3. 設定條件：
 - 指標查詢 (PromQL)
 - 閾值 (Thresholds)
 - 持續時間 (For X Minutes)
4. 選擇通知方式 (Email、Slack、Webhook)。
5. 點擊 「Apply」（應用）。

示例：如果 CPU 使用率超過 80% 發送警報

```
100 * (1 - avg(rate(node_cpu_seconds_total{mode="idle"}[5m])))
```

條件：

- 大於 80% 時觸發警報
- 持續 5 分鐘後發送通知





6. Panel 進階技巧

6.1 變數 (Variables)

Grafana 允許你使用 **變數 (Variables)** 來動態選擇數據源，例如：

```
node_memory_Active_bytes{instance="$server"}
```

用戶可以在 Dashboard 內 選擇 `\$server` 變數來查看不同主機的數據。

6.2 Panel Drilldown

你可以設定 **Panel** 的點擊連結，例如：

- 點擊 Panel 跳轉到詳細儀表板
- 點擊 Panel 打開 API 查詢頁面

設定方式：

- 進入 Panel 設定。
- 在 「Links」 頁面新增連結。
- 設定目標 URL (如 `/dashboard?var-server=xyz`)。

7. 總結

| 功能 | 說明 |
|----------------|---------------------------|
| 新增 Panel | `Add Panel`，選擇數據來源 & 查詢方式 |
| Panel 類型 | 折線圖、表格、單值、儀表盤、長條圖等 |
| Panel 設定 | 設定標題、單位、圖例、閾值 |
| 警報 (Alert) | 設定條件，發送 Email / Slack 通知 |
| 變數 (Variables) | 讓 Panel 支援動態篩選 |

透過 **Panel**，你可以 將系統監控數據視覺化，提升可讀性，並在異常時發送警報！🚀



以繁體中文詳細說明:普米、VictoriaMetrics一小時內的cpu、mem
使用率：要有平均 最大 最小



Prometheus & VictoriaMetrics 監控 CPU、記憶體使用率（平均、最大、最小）

在 **Prometheus** 或 **VictoriaMetrics (VM)** 中，監控 **1 小時內 CPU、記憶體的平均值、最大值、最小值**，可以透過 **PromQL** 查詢。

1. 監控 CPU 使用率

1.1 Prometheus 監控 `process_cpu_seconds_total`

計算 CPU 使用率 (**Prometheus**)：

```
100 * (1 - avg(rate(node_cpu_seconds_total{mode="idle"}[1h])))
```

這表示：

- `node_cpu_seconds_total`：累積的 CPU 使用時間
- `mode="idle"`：篩選閒置時間 (`idle`)，計算 **非閒置時間即 CPU 使用率**
- `rate(...[1h])`：計算 **最近 1 小時 CPU 使用率變化率**
- `1 - idle` 計算 **CPU 使用率**
- 乘 `100` 轉換為 **百分比**

1.2 VictoriaMetrics 監控 `process_cpu_seconds_total`

VictoriaMetrics 兼容 PromQL，使用方式相同：





```
100 * (1 - avg(rate(process_cpu_seconds_total{mode="idle"}[1h])))
```

1.3 計算 1 小時內 CPU 使用率的「平均、最大、最小」

```
avg_over_time(100 * (1 - rate(node_cpu_seconds_total{mode="idle"}[1h]))  
[1h])
```

- `avg_over_time(...)` 計算 1 小時內的 CPU 平均使用率。

```
max_over_time(100 * (1 - rate(node_cpu_seconds_total{mode="idle"}[1h]))  
[1h])
```

- `max_over_time(...)` 計算 1 小時內 CPU 最高使用率。

```
min_over_time(100 * (1 - rate(node_cpu_seconds_total{mode="idle"}[1h]))  
[1h])
```

- `min_over_time(...)` 計算 1 小時內 CPU 最低使用率。

2. 監控記憶體使用率

2.1 Prometheus 監控 `node_memory_Active_bytes`

```
100 * (node_memory_Active_bytes / node_memory_MemTotal_bytes)
```

這表示：

- `node_memory_Active_bytes`：當前活躍記憶體 (bytes)
- `node_memory_MemTotal_bytes`：總記憶體 (bytes)
- `Active / Total * 100` 計算記憶體使用率 (百分比)

2.2 VictoriaMetrics 監控 `process_resident_memory_bytes`





```
100 * (process_resident_memory_bytes / node_memory_MemTotal_bytes)
```

- `process_resident_memory_bytes`：VictoriaMetrics 使用的記憶體
- `MemTotal_bytes`：總記憶體
- 轉換為百分比

2.3 計算 1 小時內記憶體使用率的「平均、最大、最小」

```
avg_over_time(100 * (node_memory_Active_bytes /  
node_memory_MemTotal_bytes)[1h])
```

- `avg_over_time(...)` 計算 1 小時內記憶體的平均使用率。

```
max_over_time(100 * (node_memory_Active_bytes /  
node_memory_MemTotal_bytes)[1h])
```

- `max_over_time(...)` 計算 1 小時內記憶體最高使用率。

```
min_over_time(100 * (node_memory_Active_bytes /  
node_memory_MemTotal_bytes)[1h])
```

- `min_over_time(...)` 計算 1 小時內記憶體最低使用率。

3. Grafana 設定監控 Panel

3.1 新增 Panel

1. 進入 **Grafana**
2. 點擊 「+」 → 「Add Panel」
3. 選擇 **Prometheus**（或 **VictoriaMetrics**）作為數據來源
4. 輸入 **PromQL** 查詢





- **CPU 平均值**

```
avg_over_time(100 * (1 - rate(node_cpu_seconds_total{mode="idle"} [1h]))[1h])
```

- **CPU 最大值**

```
max_over_time(100 * (1 - rate(node_cpu_seconds_total{mode="idle"} [1h]))[1h])
```

- **CPU 最小值**

```
min_over_time(100 * (1 - rate(node_cpu_seconds_total{mode="idle"} [1h]))[1h])
```

- **記憶體平均值**

```
avg_over_time(100 * (node_memory_Active_bytes / node_memory_MemTotal_bytes)[1h])
```

- **記憶體最大值**

```
max_over_time(100 * (node_memory_Active_bytes / node_memory_MemTotal_bytes)[1h])
```

- **記憶體最小值**

```
min_over_time(100 * (node_memory_Active_bytes / node_memory_MemTotal_bytes)[1h])
```

5. 選擇視覺化方式

- 折線圖 (Graph) → 觀察趨勢
- 單值 (Stat) → 只顯示當前值
- 儀表盤 (Gauge) → 顯示百分比

6. 點擊 `Save` (儲存), 即完成設定。

4. API 查詢方式





如果需要透過 API 查詢 **VictoriaMetrics** 或 **Prometheus** 的 CPU/記憶體使用率：

```
curl "http://vmselect:8481/select/0/prometheus/api/v1/query?  
query=avg_over_time(100*(1-rate(node_cpu_seconds_total{mode='idle'})[1h]))  
[1h]"
```

這將回傳 1 小時內 **CPU 使用率的平均值**。

5. 總結

| 指標 | PromQL 查詢 |
|--------------------------|--|
| CPU 平均使用率 (1 小時內) | `avg_over_time(100 * (1 - rate(node_cpu_seconds_total{mode="idle"})[1h]))[1h]` |
| CPU 最大使用率 (1 小時內) | `max_over_time(100 * (1 - rate(node_cpu_seconds_total{mode="idle"})[1h]))[1h]` |
| CPU 最小使用率 (1 小時內) | `min_over_time(100 * (1 - rate(node_cpu_seconds_total{mode="idle"})[1h]))[1h]` |
| 記憶體平均使用率 (1 小時內) | `avg_over_time(100 * (node_memory_Active_bytes / node_memory_MemTotal_bytes)[1h])` |
| 記憶體最大使用率 (1 小時內) | `max_over_time(100 * (node_memory_Active_bytes / node_memory_MemTotal_bytes)[1h])` |
| 記憶體最小使用率 (1 小時內) | `min_over_time(100 * (node_memory_Active_bytes / node_memory_MemTotal_bytes)[1h])` |

這些查詢可以在 **Grafana 設定 Panel**，幫助你監控 **Prometheus、VictoriaMetrics** 的 CPU、記憶體 平均、最大、最小 使用率！



以繁體中文詳細說明:普米、VictoriaMetrics 的pql都一樣，這樣我怎麼知道cpu、mem的使用率來是Prometheus還是VictoriaMetrics





如何區分 Prometheus 和 VictoriaMetrics 提供的 CPU、記憶體使用率數據？

因為 Prometheus 和 VictoriaMetrics (VM) 都支持 PromQL，在 Grafana 或 `vmui` 查詢時，語法相同，因此我們需要通過 **數據來源 (Data Source)** 或 **特定 Metrics 名稱** 來區分 CPU、記憶體的數據來自哪個系統。

1. 方法 1：檢查數據來源 (Data Source)

在 Grafana 中

在 Grafana 設定 Panel 時，你可以選擇不同的 **Data Source** 來確保查詢來自 Prometheus 或 VictoriaMetrics：

1. 進入 **Grafana Dashboard**
2. 點擊 「Edit Panel」 (編輯面板)
3. 在 **Data Source (數據來源)** 選擇：
 - `Prometheus` (如果數據來自 Prometheus)
 - `VictoriaMetrics` (如果數據來自 VictoriaMetrics)
4. 輸入 PromQL 查詢：

```
rate(process_cpu_seconds_total[5m])
```

5. 比較兩個 **Data Source** 返回的數據是否一致

這是最直接的方法，確保查詢來自哪個系統。

2. 方法 2：檢查 `job` 或 `instance` 標籤

Prometheus 和 VictoriaMetrics 收集的 Metrics 來源不同，可以透過 `job` 或 `instance` 來區分。





2.1 查詢 Prometheus 來源

```
rate(process_cpu_seconds_total{job="prometheus"}[5m])
```

- 如果這個查詢有數據，代表 **CPU 使用率來自 Prometheus**。
- `job="prometheus"` 可能是 **Prometheus 自身的監控數據**。

2.2 查詢 VictoriaMetrics 來源

```
rate(process_cpu_seconds_total{job="victoriametrics"}[5m])
```

- 如果這個查詢有數據，代表 **CPU 使用率來自 VictoriaMetrics**。
- `job="victoriametrics"` 代表 **VictoriaMetrics 自身的監控數據**。

這個方法適用於：

- 兩個系統都在同一個 **Grafana 環境**，但你希望區分數據來源。

3. 方法 3：查詢 `prometheus_tsdb_head_series` 或 `vm_data_size_bytes`

Prometheus 和 VictoriaMetrics 內部的存儲指標不同，可以用來確保查詢來源。

3.1 查詢 Prometheus 存儲的時間序列數量

```
prometheus_tsdb_head_series
```

- 如果這條指標有數據，代表 **查詢的是 Prometheus 的存儲狀態**。

3.2 查詢 VictoriaMetrics 存儲的數據大小

```
vm_data_size_bytes
```





- 如果這條指標有數據，代表 **查詢的是 VictoriaMetrics 的存儲狀態**。

這個方法適用於：

- 確認你查詢的數據來自哪個時序數據庫（**TSDB**）。

4. 方法 4：查詢 `scrape_samples_scraped`

Prometheus 和 VictoriaMetrics 都有 `scrape_samples_scraped` 指標，但標籤不同，可以用來區分。

4.1 Prometheus 的 `scrape_samples_scraped`

```
rate(scrape_samples_scraped{job="prometheus"}[5m])
```

- 代表 **Prometheus 收集的樣本數據**。

4.2 VictoriaMetrics 的 `scrape_samples_scraped`

```
rate(scrape_samples_scraped{job="victoriametrics"}[5m])
```

- 代表 **VictoriaMetrics 收集的樣本數據**。

這個方法適用於：

- 確認是哪個系統負責抓取數據。

5. 方法 5：API 直接查詢

如果你可以直接訪問 Prometheus 或 VictoriaMetrics，你可以透過 API 來查詢數據，並比對返回的結果。





5.1 查詢 Prometheus

```
curl "http://prometheus:9090/api/v1/query?  
query=rate(process_cpu_seconds_total[5m])"
```

如果返回數據，表示 **CPU 使用率來自 Prometheus。**

5.2 查詢 VictoriaMetrics

```
curl "http://vmselect:8481/select/0/prometheus/api/v1/query?  
query=rate(process_cpu_seconds_total[5m])"
```

如果返回數據，表示 **CPU 使用率來自 VictoriaMetrics。**

這個方法適用於：

- 直接從 API 端確保數據來源。

6. 方法 6：使用不同的 Data Source Prefix

如果 Prometheus 和 VictoriaMetrics 都在同一個 Grafana，你可以 手動標記數據 來區分來源，例如：

```
scrape_configs:  
  - job_name: "prometheus_metrics"  
    static_configs:  
      - targets: ["prometheus:9090"]  
  
  - job_name: "victoriametrics_metrics"  
    static_configs:  
      - targets: ["vmselect:8481"]
```

然後在 Grafana 查詢：

```
rate(process_cpu_seconds_total{job="prometheus_metrics"}[5m])
```





或

```
rate(process_cpu_seconds_total{job="victoriameetrics_metrics"}[5m])
```

✓ 這個方法適用於：

- 管理多個數據來源時，手動標記來源。

7. 總結

| 方法 | 用途 | 適用場景 |
|---|---|---------------|
| 方法 1：檢查 Data Source | 透過 Grafana 選擇 `Prometheus` 或 `VictoriaMetrics` | 適用於 Grafana |
| 方法 2：查詢 `job` 或 `instance` | 透過 `job="prometheus"` 或 `job="victoriameetrics"` 來區分 | 適用於 PromQL 查詢 |
| 方法 3：查詢 Prometheus/VictoriaMetrics 的 TSDB 指標 | `prometheus_tsdb_head_series` VS. `vm_data_size_bytes` | 確認數據來自哪個 TSDB |
| 方法 4：查詢 `scrape_samples_scraped` | `scrape_samples_scraped{job="prometheus"}` VS. `scrape_samples_scraped{job="victoriameetrics"}` | 確認數據是由哪個系統收集的 |
| 方法 5：使用 API 查詢 | `curl "http://prometheus:9090/api/v1/query"` VS. `curl "http://vmselect:8481/select/0/prometheus/api/v1/query"` | 適用於後端 API 查詢 |
| 方法 6：使用不同的 `job` 前綴 | `job="prometheus_metrics"` VS. `job="victoriameetrics_metrics"` | 適用於手動標記來源 |

如果你 只是在 Grafana 查詢數據，最簡單的方法是：

- 在 Panel 設定 Data Source
- 查詢 `job` 標籤





3. 使用 API 查詢來比對

這樣，你就可以 區分 CPU、記憶體使用率的數據是來自 **Prometheus** 還是 **VictoriaMetrics** ! 🚀



以繁體中文詳細說明:Prometheus、VictoriaMetrics每筆數據的寫入 時間、每秒寫入 時間幾個byte數據



Prometheus & VictoriaMetrics：計算寫入、查詢時間及數據量

在 **Prometheus** 和 **VictoriaMetrics (VM)** 中，我們可以透過 **PromQL 查詢** 來分析：

1. 每筆數據的寫入時間
2. 每筆數據的查詢時間
3. 每秒寫入多少數據 (bytes)
4. 每秒查詢多少數據 (bytes)

這些數據可以用來評估 系統效能，並比較 **Prometheus** 與 **VictoriaMetrics** 在不同場景下的表現。

1. 計算每筆數據的「寫入時間」

1.1 Prometheus 計算寫入時間

```
rate(prometheus_tsdb_head_samples_appended_total[5m])
```

- `prometheus_tsdb_head_samples_appended_total` : 記錄 **Prometheus TSDB** 每秒寫入的樣本數





- `rate(...[5m])`：計算最近 5 分鐘內 每秒寫入的數據筆數
- 數值單位：每秒寫入多少行數據 (samples/s)

1.2 VictoriaMetrics 計算寫入時間

```
rate(vm_rows_inserted_total[5m])
```

- `vm_rows_inserted_total`：VictoriaMetrics 每秒寫入的數據行數
- `rate(...[5m])`：計算最近 5 分鐘內 VictoriaMetrics 每秒寫入的數據行數

2. 計算每筆數據的「查詢時間」

2.1 Prometheus 計算查詢時間

```
rate(prometheus_http_request_duration_seconds_sum{handler="/api/v1/query"}[5m]) /  
rate(prometheus_http_request_duration_seconds_count{handler="/api/v1/query"}[5m])
```

- `prometheus_http_request_duration_seconds_sum`：累積的 HTTP 查詢時間 (秒)
- `prometheus_http_request_duration_seconds_count`：查詢請求數量
- 結果單位：每筆查詢的平均時間 (秒/查詢)

2.2 VictoriaMetrics 計算查詢時間

```
rate(vm_select_duration_seconds_sum[5m]) /  
rate(vm_select_duration_seconds_count[5m])
```

- `vm_select_duration_seconds_sum`：VictoriaMetrics 累積的查詢時間
- `vm_select_duration_seconds_count`：VictoriaMetrics 查詢請求的總次數
- 結果單位：每筆查詢的平均時間 (秒/查詢)





3. 計算每秒寫入的數據量 (bytes/s)

3.1 Prometheus 計算每秒寫入的數據量

```
rate(prometheus_tsdb_storage_blocks_bytes[5m])
```

- `prometheus_tsdb_storage_blocks_bytes` : Prometheus 當前存儲的 **磁碟空間 (bytes)**
- `rate(...[5m])` : 計算最近 5 分鐘內 **每秒新增多少 bytes**
- **單位 : bytes/s**

3.2 VictoriaMetrics 計算每秒寫入的數據量

```
rate(vm_data_size_bytes[5m])
```

- `vm_data_size_bytes` : VictoriaMetrics 存儲的數據量 (bytes)
- `rate(...[5m])` : 計算 **每秒寫入多少 bytes**
- **單位 : bytes/s**

4. 計算每秒查詢的數據量 (bytes/s)

4.1 Prometheus 計算查詢數據量

```
rate(prometheus_http_request_duration_seconds_sum{handler="/api/v1/query_range"}[5m])
```

- `prometheus_http_request_duration_seconds_sum` : 累積的查詢時間
- `rate(...[5m])` : 計算最近 5 分鐘內 **查詢 API 每秒處理多少數據**
- **單位 : bytes/s**

4.2 VictoriaMetrics 計算查詢數據量





```
rate(vm_cache_requests_size_bytes[5m])
```

- `vm_cache_requests_size_bytes` : VictoriaMetrics 查詢請求的總數據量 (bytes)
- `rate(...[5m])` : 計算最近 5 分鐘內 **每秒查詢多少 bytes**
- **單位 : bytes/s**

5. 設定 Grafana 監控面板

步驟 1：新增 Panel

1. 打開 Grafana
2. 點擊 `+` → `Add Panel`
3. 選擇 Data Source (Prometheus 或 VictoriaMetrics)
4. 輸入 PromQL 查詢

步驟 2：輸入查詢

- 寫入時間 (Prometheus)

```
rate(prometheus_tsdb_head_samples_appended_total[5m])
```

- 寫入時間 (VictoriaMetrics)

```
rate(vm_rows_inserted_total[5m])
```

- 檢查時間 (Prometheus)

```
rate(prometheus_http_request_duration_seconds_sum{handler="/api/v1/query"}[5m]) /  
rate(prometheus_http_request_duration_seconds_count{handler="/api/v1/query"}[5m])
```

- 檢查時間 (VictoriaMetrics)

```
rate(vm_select_duration_seconds_sum[5m]) /  
rate(vm_select_duration_seconds_count[5m])
```





- 每秒寫入數據量 (**Prometheus**)

```
rate(prometheus_tsdb_storage_blocks_bytes[5m])
```

- 每秒寫入數據量 (**VictoriaMetrics**)

```
rate(vm_data_size_bytes[5m])
```

- 每秒查詢數據量 (**Prometheus**)

```
rate(prometheus_http_request_duration_seconds_sum{handler="/api/v1/query_range"}[5m])
```

- 每秒查詢數據量 (**VictoriaMetrics**)

```
rate(vm_cache_requests_size_bytes[5m])
```

步驟 3：選擇視覺化類型

- **折線圖 (Graph)** → 觀察趨勢
- **單值 (Stat)** → 顯示當前數據
- **儀表盤 (Gauge)** → 顯示百分比或數據量

步驟 4：儲存設定

1. 點擊 `Save` (儲存)
2. Panel 會即時顯示寫入、查詢的數據量與時間。

6. 總結

| 指標 | PromQL 查詢 |
|-----------------------------|---|
| 每秒寫入筆數 (Prometheus) | `rate(prometheus_tsdb_head_samples_appended_total[5m])` |
| 每秒寫入筆數 (VictoriaMetrics) | `rate(vm_rows_inserted_total[5m])` |





| 指標 | PromQL 查詢 |
|-------------------------------|---|
| 每筆查詢平均時間 (Prometheus) | `rate(prometheus_http_request_duration_seconds_sum{handler="/api/v1/query"}[5m]) / rate(prometheus_http_request_duration_seconds_count{handler="/api/v1/query"}[5m])` |
| 每筆查詢平均時間 (VictoriaMetrics) | `rate(vm_select_duration_seconds_sum[5m]) / rate(vm_select_duration_seconds_count[5m])` |
| 每秒寫入數據量 (Prometheus) | `rate(prometheus_tsdb_storage_blocks_bytes[5m])` |
| 每秒寫入數據量 (VictoriaMetrics) | `rate(vm_data_size_bytes[5m])` |
| 每秒查詢數據量 (Prometheus) | `rate(prometheus_http_request_duration_seconds_sum{handler="/api/v1/query_range"}[5m])` |
| 每秒查詢數據量 (VictoriaMetrics) | `rate(vm_cache_requests_size_bytes[5m])` |

透過這些 PromQL 查詢，你可以 監控 Prometheus 和 VictoriaMetrics 的寫入、查詢時間，以及數據吞吐量 (bytes/s)，並在 Grafana 上建立視覺化監控！🚀



以繁體中文詳細說明:VictoriaMetrics、vmstorage 儲存的byte、筆數、row數



VictoriaMetrics / `vmstorage` 儲存的 byte 數、筆數、row 數查詢

在 VictoriaMetrics (VM) 中，`vmstorage` 負責數據的存儲，並提供多種指標來監控數據量，包括：

1. 儲存的 Byte 數（存儲使用量）
2. 儲存的數據筆數（寫入的樣本數量）
3. 儲存的 Row 數（時序數據行數）

這些指標可以透過 PromQL 查詢，並用 Grafana 或 `vmui` 進行監控。





1. 查詢 VictoriaMetrics `vmstorage` 儲存的 Byte 數

PromQL 查詢

```
vm_data_size_bytes
```

- 這個指標表示 **VictoriaMetrics** 當前存儲的數據量 (bytes)。
- 你可以用這個指標來監控 **磁碟使用量**。

變化率 (過去 1 小時增加了多少數據量)

```
increase(vm_data_size_bytes[1h])
```

- 這將返回 **過去 1 小時內存儲數據的增量 (bytes)**。
- 你可以用來分析 **數據增長速率**。

2. 查詢 VictoriaMetrics `vmstorage` 儲存的筆數

PromQL 查詢

```
vm_rows
```

- 這個指標表示 **VictoriaMetrics** 當前存儲的「時序數據行數」 (**Rows**)。
- 每一行對應於一個 **時間點的監測數據**。

變化率 (過去 1 小時寫入了多少筆數據)

```
increase(vm_rows[1h])
```

- 這表示 **過去 1 小時內, VictoriaMetrics 寫入了多少筆數據**。





- 如果這個值很高，表示 系統存儲的數據量正在快速增長。

3. 查詢 VictoriaMetrics `vmstorage` 儲存的 row 數

在 VictoriaMetrics 中，`row`（行數）代表存儲的時序數據，每行包含：

- 指標名稱 (Metric Name)
- 標籤 (Labels)
- 時間戳 (Timestamp)
- 值 (Value)

PromQL 查詢

```
vm_rows_inserted_total
```

- 這個指標表示 VictoriaMetrics 總共寫入的 row 數（累積值）。
- 它是一個 Counter（計數器），只會遞增。

每秒寫入的 Row 數

```
rate(vm_rows_inserted_total[1m])
```

- 這表示 VictoriaMetrics 每秒寫入的數據行數 (rows/s)，用來監控寫入速率。

每小時寫入的 Row 數

```
increase(vm_rows_inserted_total[1h])
```

- 這表示 過去 1 小時內寫入的總數據行數。





4. 設定 Grafana 監控面板

如果你想在 Grafana 監控 VictoriaMetrics 儲存的 byte 數、筆數、row 數，請按照以下步驟：

步驟 1：新增 Panel

1. 打開 Grafana
2. 點擊 `+` → `Add Panel`
3. 選擇 Data Source (VictoriaMetrics)
4. 輸入 PromQL 查詢

步驟 2：輸入查詢

| 監控項目 | PromQL 查詢 |
|--------------------|--|
| 當前存儲的 Byte 數 | `vm_data_size_bytes` |
| 過去 1 小時內存儲的 Byte 數 | `increase(vm_data_size_bytes[1h])` |
| 當前存儲的 Row 數 | `vm_rows` |
| 過去 1 小時內存儲的 Row 數 | `increase(vm_rows[1h])` |
| 總寫入的 Row 數 (累計) | `vm_rows_inserted_total` |
| 每秒寫入的 Row 數 | `rate(vm_rows_inserted_total[1m])` |
| 每小時寫入的 Row 數 | `increase(vm_rows_inserted_total[1h])` |

步驟 3：選擇視覺化類型

- **折線圖 (Graph)** → 觀察趨勢
- **單值 (Stat)** → 顯示當前數據
- **儀表盤 (Gauge)** → 顯示百分比或數據量

步驟 4：儲存設定

1. 點擊 `Save` (儲存)
2. Panel 會即時顯示 VictoriaMetrics 的存儲數據





5. 總結

| 監控項目 | PromQL 查詢 |
|--------------------|--|
| 當前存儲的 Byte 數 | `vm_data_size_bytes` |
| 過去 1 小時內存儲的 Byte 數 | `increase(vm_data_size_bytes[1h])` |
| 當前存儲的 Row 數 | `vm_rows` |
| 過去 1 小時內存儲的 Row 數 | `increase(vm_rows[1h])` |
| 總寫入的 Row 數（累計） | `vm_rows_inserted_total` |
| 每秒寫入的 Row 數 | `rate(vm_rows_inserted_total[1m])` |
| 每小時寫入的 Row 數 | `increase(vm_rows_inserted_total[1h])` |

透過這些 PromQL 查詢，你可以：

- 監控 VictoriaMetrics 儲存的數據大小
- 監測過去 1 小時內的存儲增長情況
- 分析每秒、每小時寫入的數據量
- 優化 VictoriaMetrics `vmstorage` 的存儲效能

這樣，你就可以透過 Grafana 或 `vmui` 來分析 VictoriaMetrics 的數據存儲行為！🚀



以繁體中文詳細說明：為什麼我的Grafana panel，常會出現同樣的stat兩次，很影響觀看



Grafana Panel 出現重複 Stat 值的可能原因與解決方法

當你的 Grafana Panel 顯示 相同的數據 (Stat) 重複兩次或多次，可能是因為：





1. PromQL 查詢結果包含多個 Series
 2. Query 沒有適當聚合
 3. Panel 顯示格式問題
 4. 變數 (Variables) 重複選取
 5. Prometheus 或 VictoriaMetrics 的 `job`、`instance` 沒有正確篩選
-

1. 問題 1：PromQL 查詢結果包含多個 Series

問題描述

當你使用 **Stat Panel** 時，如果 PromQL 查詢返回 **多個 Series**（不同 **instance** 或 **job**），Grafana 會 **自動將這些 Series 都顯示出來**，導致數據重複。

解決方案

加入 `sum()` 聚合函數，確保只返回一個數據

```
sum(vm_rows)
```

- `sum(vm_rows)`：將所有 `vm_rows` 數據 **合併為一個總數**，避免多個值出現。

加入 `by()` 篩選，只顯示單個 `job` 或 `instance`

```
sum(vm_rows) by (job)
```

- 這會讓 **Grafana** 只顯示不同 `job` 的數據，不會重複。
-

2. 問題 2：Query 沒有適當聚合

問題描述





如果 PromQL 沒有適當的 **聚合函數**，不同 `instance` 可能會返回多個值，導致 Panel 出現重複的數據。

解決方案

確保只返回單個數值

```
avg(vm_rows)
```

或

```
max(vm_rows)
```

這樣 Grafana 只會顯示 **單一數值**，避免重複。

3. 問題 3：Panel 設定問題

問題描述

在 **Grafana Panel 設定** 中，如果「Stat」的 **顯示模式** 設定錯誤，可能會顯示多個相同數據。

解決方案

步驟 1：檢查 `Reduce options`

1. 進入 **Panel**
2. 點擊 「**Edit**」
3. 選擇 「**Reduce options**」
4. 確保選擇：
 - 「**Last**」（最後一個值）
 - 或 「**Mean**」（平均值）

步驟 2：檢查 `Legend`（圖例）





1. 在 「Panel settings」 → 「Legend」
 2. 選擇 「Hide duplicate labels」
 3. 確保 只顯示你需要的數據
-

4. 問題 4：變數（Variables）重複選取

問題描述

如果你的 **Grafana Dashboard** 使用 **Variables**（變數），並且變數選擇了 多個值，Stat Panel 可能會 一次顯示多組相同數據。

解決方案

1. 進入 **Dashboard** 設定
2. 點擊 「**Variables**」
3. 檢查 變數是否選擇了多個值
4. 如果是多個值，改成 單一選擇

如果一定要顯示多個值，可以用 `sum()` 合併

```
sum(vm_rows) by (job)
```

5. 問題 5：Prometheus 或 VictoriaMetrics 的 `job`、`instance` 沒有正確篩選

問題描述

如果 你的查詢沒有適當的 `job` 或 `instance` 過濾條件，Prometheus / VictoriaMetrics 可能會返回相同的數據，但來自不同 **instance**，導致 Grafana 顯示重複的 Stat。

解決方案





加上 `instance` 或 `job` 篩選

```
sum(vm_rows) by (instance)
```

- 如果只需要某個 **instance**, 則加上 `instance="xxx"`

```
sum(vm_rows) by (instance) | instance="vmstorage-1"
```

或加上 `job` 篩選

```
sum(vm_rows) by (job) | job="victoriametrics"
```

6. 總結

| 問題 | 解決方案 |
|--|----------------------------------|
| PromQL 查詢結果包含多個 Series | 使用 `sum()`、`avg()`、`max()` 聚合 |
| Query 沒有適當聚合 | 確保查詢只返回一個數值 |
| Panel 設定問題 | 設定 `Reduce options` 為 `Last` |
| 變數 (Variables) 重複選取 | 限制變數為單一值 |
| Prometheus / VM 返回重複的 `job` / `instance` | 加入 `by(job)` 或 `by(instance)` 篩選 |

✓ 最佳解法 (推薦使用)

```
sum(vm_rows) by (job)
```

這樣 **Stat Panel** 只會顯示一個值，避免重複數據！

這樣，你就可以讓 **Grafana Panel** 只顯示正確的數據，避免影響觀看！🚀





以繁體中文詳細說明:vmagent ha example



VictoriaMetrics `vmagent` 高可用 (HA) 架構示例

VictoriaMetrics 的 `vmagent` 是一個高效能的 **Prometheus** 兼容數據抓取代理，主要用於：

1. 從各種資料來源 (**Prometheus targets**、**Pushgateway** 等) 抓取 Metrics
2. 將數據轉發到 `vminsert` (**VictoriaMetrics**) 或其他 TSDB
3. 支援負載均衡和高可用 (HA) 部署

在 **高可用 (HA)** 架構下，通常會部署 **多個 `vmagent` 節點**，確保：

- 單點故障不影響數據收集
- 均衡負載，提升性能
- 確保數據持續可用

1. `vmagent` HA 架構設計

在 HA 部署下，我們通常會有：

- 多個 `vmagent` 節點 (兩個或更多)
- 使用 `vmagent` HA 標籤 (`replica`)
- 後端 `vminsert` 負責數據存儲
- 可以使用 `load balancer` 來均衡 `vmagent` 負載

架構示意圖





在這個架構中：

- 多個 `vmagent` (**HA 模式**) 會從相同的 `scrape_configs` 來源抓取數據
- 使用 `replica` 標籤區分不同的 `vmagent`
- 數據會被寫入 `vminsert`，並最終存儲到 `vmstorage`
- `vmselect` 可用來查詢數據

2. `vmagent` HA 部署的 `docker-compose.yml` 示例

步驟 1：建立 HA `vmagent` 配置

```
version: '3.7'  
services:
```





```
vmagent-1:
  image: victoriametrics/vmagent
  container_name: vmagent-1
  restart: always
  volumes:
    - ./scrape_config.yml:/etc/vmagent.yml
  command:
    - "-promscrape.config=/etc/vmagent.yml"
    - "-remoteWrite.url=http://vminsert:8480/insert/"
    - "-remoteWrite.label=replica=a" # 設定 HA 標籤
  ports:
    - "8429:8429"
  networks:
    - monitoring

vmagent-2:
  image: victoriametrics/vmagent
  container_name: vmagent-2
  restart: always
  volumes:
    - ./scrape_config.yml:/etc/vmagent.yml
  command:
    - "-promscrape.config=/etc/vmagent.yml"
    - "-remoteWrite.url=http://vminsert:8480/insert/"
    - "-remoteWrite.label=replica=b" # 設定 HA 標籤
  ports:
    - "8430:8429"
  networks:
    - monitoring

vminsert:
  image: victoriametrics/vminsert
  container_name: vminsert
  restart: always
  command: -storageNode=vmstorage:8400
  ports:
    - "8480:8480"
  networks:
    - monitoring

vmstorage:
  image: victoriametrics/vmstorage
  container_name: vmstorage
  restart: always
```





```
ports:
  - "8428:8428"
networks:
  - monitoring

vmselect:
  image: victoriametrics/vmselect
  container_name: vmselect
  restart: always
  command: -storageNode=vmstorage:8400
  ports:
    - "8481:8481"
networks:
  - monitoring

networks:
  monitoring:
```

在這個 `docker-compose.yml` 配置中：

- `vmagent-1` 和 `vmagent-2` 會同時從 `scrape_config.yml` 收集數據
- 它們會將數據推送到 `vminsert`
- 每個 `vmagent` 都有 `--remoteWrite.label=replica=x` 來標記不同副本

3. `vmagent` 高可用的 `scrape_config.yml` 示例

步驟 2：配置 `scrape_config.yml`，確保所有 `vmagent` 都使用相同的配置

```
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: "node_exporter"
    static_configs:
      - targets: ["node-exporter-1:9100", "node-exporter-2:9100"]

  - job_name: "app"
```





```
static_configs:
  - targets: ["app-server-1:8000", "app-server-2:8000"]
```

這個配置確保：

- 兩個 `vmagent` 都從相同的 `targets` 抓取數據
- 即使一個 `vmagent` 掛掉，另一個仍然能夠運行

4. 查詢 `vmagent` HA 數據

在 **VictoriaMetrics** / `vmui` 或 **Grafana** 中，可以使用 PromQL 查詢：

```
sum by (replica) (rate(vm_rows_inserted_total[5m]))
```

- 這會顯示 **不同 `vmagent` 副本 (replica)** 所寫入的數據
- 如果某個 `vmagent` 掛掉，則該 `replica` 的數據會消失

如果你想查看 **單一 `vmagent` 寫入的數據量**：

```
rate(vm_rows_inserted_total{replica="a"}[5m])
```

這樣可以確認 **HA 機制是否正常運作**。

5. 監控 `vmagent` HA 狀態

你可以在 **Grafana** 設置 **HA 監控面板**：

1. **每個 `vmagent` 的寫入速率**

```
sum(rate(vm_rows_inserted_total[5m])) by (replica)
```

2. **每個 `vmagent` 的內存使用量**





```
process_resident_memory_bytes{job="vmagent"}
```

3. 檢查是否有 `vmagent` 宿機

```
up{job="vmagent"}
```

6. 總結

| 功能 | HA 設計 |
|---------------|--|
| 高可用 `vmagent` | 部署多個 `vmagent` 並設置 `-remoteWrite.label=replica=x` |
| 負載均衡 | `vmagent` 負責拉取相同數據，負載分散 |
| 查詢 HA 數據 | `sum(rate(vm_rows_inserted_total[5m])) by (replica)` |
| 監控 HA 狀態 | `up{job="vmagent"}` 確保 `vmagent` 存活 |

透過這樣的部署，你可以確保：

- 當一個 `vmagent` 掛掉，另一個仍然能夠持續收集數據
- 提升 `vmagent` 的負載均衡
- 確保數據收集不間斷
- 在 Grafana 監控 `vmagent` HA 狀態

這樣，你就能建立 可靠的 VictoriaMetrics `vmagent` 高可用架構！🚀



以繁體中文詳細說明:vmagent-1、vmagent-2是把資料收進2份，還是加起來只收一份，global:
scrape_interval: 15s

```
scrape_configs:  
  - job_name: "node_exporter"  
    static_configs:  
      - targets: ["node-exporter-1:9100", "node-exporter-2:9100"]
```



2:9100"]

```
- job_name: "app"
  static_configs:
    - targets: ["app-server-1:8000", "app-server-2:8000"]
```



vmagent-1、vmagent-2 是收兩份還是只收一份？

當 `vmagent-1` 和 `vmagent-2` 部署在 HA（高可用）模式下，並且它們使用相同的 `scrape_configs.yml` 來收集數據，有兩種可能的情況：

1. 兩個 `vmagent` 都抓取相同的 `targets`，最終存入 2 份相同數據（重複寫入）
2. 如果 `remote_write.label=replica=X`，VictoriaMetrics 會將 `replica` 標籤去重，保留 1 份數據

結論

- 如果 `vmagent-1` 和 `vmagent-2` 只是獨立運行（未使用 `deduplication` 去重機制），則它們會各自抓取相同的數據並寫入 2 份相同的數據。
- 如果 `remoteWrite.label=replica=x`，並且 `vmstorage` 啟用了去重機制 (`dedup.minScrapeInterval`)，則 VictoriaMetrics 會自動去重，確保只存一份數據。

1. `vmagent` HA 模式下的行為

情境 1：沒有 `replica` 標籤（會重複存入 2 份數據）

如果 `vmagent-1` 和 `vmagent-2` 沒有 `replica` 標籤，則：

- `vmagent-1` 抓取相同的 `targets`
 - `vmagent-2` 也抓取相同的 `targets`
 - 結果：寫入 `vminsert` 的數據是 2 份重複數據
- ◆ PromQL 查詢時會發現同一個 `instance` 有 2 份數據





```
rate(node_cpu_seconds_total[5m])
```

這時候查詢會返回 **兩組相同的數據**。

解決方案

為了避免數據重複寫入，可以：

1. 使用 `replica` 標籤來標記不同的 `vmagent`
2. 讓 **VictoriaMetrics** 進行去重 (`deduplication`)

2. 使用 `replica` 標籤來確保去重

情境 2：使用 `replica` 來標記 `vmagent`（VictoriaMetrics 會去重）

在 `docker-compose.yml` 中，每個 `vmagent` 配置 `-remoteWrite.label=replica=x`：

```
vmagent-1:  
  image: victoriametrics/vmagent  
  command:  
    - "-promscrape.config=/etc/vmagent.yml"  
    - "-remoteWrite.url=http://vminsert:8480/insert/"  
    - "-remoteWrite.label=replica=a" # 這個 vmagent 會被標記為 replica=a  
  
vmagent-2:  
  image: victoriametrics/vmagent  
  command:  
    - "-promscrape.config=/etc/vmagent.yml"  
    - "-remoteWrite.url=http://vminsert:8480/insert/"  
    - "-remoteWrite.label=replica=b" # 這個 vmagent 會被標記為 replica=b
```

VictoriaMetrics 自動去重

- 當 **VictoriaMetrics** 啟用 `dedup.minScrapeInterval`，則：





- 即使 `vmagent-1` 和 `vmagent-2` 抓取相同的數據，VictoriaMetrics 只會存 **1 份** 數據
- `replica` 標籤在查詢時會自動去重

◆ 去重查詢

```
rate(node_cpu_seconds_total[5m])
```

- 只會返回 **一組數據（已去重）**。
- 如果要顯示 `replica` 標籤：

```
rate(node_cpu_seconds_total[5m]) by (replica)
```

3. 啟用 VictoriaMetrics 去重 (Deduplication)

如果 `vmagent-1` 和 `vmagent-2` 使用 `replica` 標籤，但 **VictoriaMetrics 未啟用 `deduplication`**，仍然可能會存 2 份數據，因此需要在 `vmselect` 啟用去重。

啟用 `deduplication`

在 `vmselect` 設定：

```
vmselect:  
  image: victoriametrics/vmselect  
  command:  
    - "-storageNode=vmstorage:8400"  
    - "-dedup.minScrapeInterval=15s" # 啟用去重，每 15s 內相同數據只存 1 份
```

這樣，VictoriaMetrics 會：

- 根據 `replica` 標籤去重
- 確保相同數據只存 **1 份**





4. 如何檢查 `vmagent` HA 是否正確運行？

檢查 `vmagent-1` 和 `vmagent-2` 是否在收集數據

```
sum by (replica) (rate(vm_rows_inserted_total[5m]))
```

- 如果 `replica` 設定正確，你會看到：

```
{replica="a"} → 20000  
{replica="b"} → 20000
```

- 但如果 VictoriaMetrics 啟用了去重，你只會看到：

```
20000 # 只存 1 份數據
```

檢查 `vmagent` 是否正常上線

```
up{job="vmagent"}
```

- 這會顯示 `vmagent-1` 和 `vmagent-2` 是否存活

檢查 `deduplication` 是否生效

```
rate(node_cpu_seconds_total[5m]) by (replica)
```

- 如果 VictoriaMetrics 啟用了去重，這條查詢會 **只顯示一組數據**

5. 總結

| 狀況 | 數據存儲行為 | 解決方案 |
|--|-------------------------|-----------------------------------|
| `vmagent-1` & `vmagent-2` 沒有 `replica` | 每個 `vmagent` 寫入相同數據 2 份 | 加上 `-remoteWrite.label=replica=x` |





| 狀況 | 數據存儲行為 | 解決方案 |
|---|--------------------------|--|
| `vmagent-1` & `vmagent-2` 有 `replica`，但未啟用去重 | VictoriaMetrics 仍存 2 份數據 | 啟用 `-dedup.minScrapeInterval=15s` |
| `vmagent-1` & `vmagent-2` 有 `replica`，並啟用 `deduplication` | 只存 1 份數據（去重成功） | 確保 `vmselect` 啟用 `--dedup.minScrapeInterval=15s` |

如果 **不想要重複數據**，請確保：

1. `vmagent-1` 和 `vmagent-2` 使用 `-remoteWrite.label=replica=x`
2. `vmselect` 啟用 `--dedup.minScrapeInterval=15s`

這樣 VictoriaMetrics 會自動去重，確保只存 1 份數據！🚀



vmagent High-Availability Exa...
PDF

以繁體中文詳細說明:vmagent High-Availability Examples.pdf



VictoriaMetrics `vmagent` 高可用 (High-Availability) 配置詳解

文件 **《vmagent High-Availability Examples.pdf》** 提供了 `vmagent` 在高可用架構下的部署方式，主要涵蓋 Push 模式和 Pull 模式 的 HA 設計。

1. `vmagent` 高可用 (HA) 部署模式

`vmagent` 可以同時支援 Pull 和 Push 模式，高可用設計確保：

- 當某個 `vmagent` 節點掛掉時，仍然能夠收集數據
- 支援負載均衡，提升整體效能
- 避免數據丟失





- 在 VictoriaMetrics 進行數據去重，確保只存一份數據

2. `vmagent` Push 模式的高可用

架構說明

在 Push 模式下：

1. 多個來源（InfluxDB、Graphite、Prometheus Remote Write 等）可以將數據推送到兩組相同配置的 `vmagent` 節點。
2. `vmagent` 節點會將數據轉發到 **VictoriaMetrics**（單節點或叢集模式）。
3. VictoriaMetrics 需要開啟數據去重（Deduplication），確保相同的數據只存一份。

架構圖



重要配置





在 `vmagent` HA 配置中，每個 `vmagent` 節點會有不同的 `replica` 標籤：

```
vmagent-1:  
  image: victoriametrics/vmagent  
  command:  
    - "-remoteWrite.url=http://vminsert:8480/insert/"  
    - "-remoteWrite.label=replica=a" # 這個 vmagent 會被標記為 replica=a  
  
vmagent-2:  
  image: victoriametrics/vmagent  
  command:  
    - "-remoteWrite.url=http://vminsert:8480/insert/"  
    - "-remoteWrite.label=replica=b" # 這個 vmagent 會被標記為 replica=b
```

◆ 如何確保不存入兩份數據？

- **VictoriaMetrics 需要開啟去重機制 (Deduplication)**
- 在 `vmselect` 配置 `--dedup.minScrapeInterval=15s`，VictoriaMetrics 會根據 `replica` 標籤去重

◆ 如何查詢 HA 數據？

```
sum by (replica) (rate(vm_rows_inserted_total[5m]))
```

這樣可以檢查不同 `vmagent` 副本的寫入情況。

3. `vmagent` Pull 模式的高可用（小到中型負載）

架構說明

- 兩個 `vmagent` 實例 從相同的目標拉取數據，然後發送到 VictoriaMetrics。
- 如果其中一個 `vmagent` 掛掉，另一個 `vmagent` 仍然可以抓取數據，避免數據丟失。
- **VictoriaMetrics 需要開啟數據去重**，避免同樣的數據被存兩次。

架構圖





◆ Pull 模式 HA 的 PromQL 查詢

```
sum(rate(vm_rows_inserted_total[5m])) by (replica)
```

這樣可以顯示 `vmagent-1` 和 `vmagent-2` 的寫入狀態。

4. `vmagent` Pull 模式的高可用 (大型負載)

當 單一 `vmagent` 無法處理所有 `scrape` 目標 時，可以透過 多個 `vmagent` 分攤 `scrape` 負載，提升效能。

架構說明

- 使用 `-promscrape.cluster.membersCount` 來設定 **有多少個 `vmagent`**
- 每個 `vmagent` 設定 `-promscrape.cluster.memberNum`，分配不同的 `scrape` 目標
- 確保每個 `vmagent` 負責不同的 `scrape` 目標，避免重複拉取

架構圖





重要參數

在 `vmagent` 啟動時，需要加上：

```
vmaagent-1:  
  command:  
    - "-promscrape.cluster.membersCount=2"  
    - "-promscrape.cluster.memberNum=0"  
vmaagent-2:  
  command:  
    - "-promscrape.cluster.membersCount=2"  
    - "-promscrape.cluster.memberNum=1"
```

這樣：

- `vmaagent-1` 只負責 **一部分的** `scrape` 目標
- `vmaagent-2` 只負責 **另一部分的** `scrape` 目標
- **減少單一 `vmaagent` 的負載，提升效能**

5. 總結





| 模式 | 優點 | 注意事項 |
|----------------------|--|--|
| Push HA | 支援多個數據來源 (InfluxDB、Graphite、Prometheus Remote Write) | 需要 `--remoteWrite.label=replica=x` 確保去重 |
| Pull HA (小負載) | 兩個 `vmagent` 抓取相同 `targets`，避免數據丟失 | VictoriaMetrics 需啟用 `--dedup.minScrapeInterval=15s` |
| Pull HA (大負載) | `vmagent` 分攤 `scrape` 目標，減少單一節點負擔 | `-promscrape.cluster.membersCount` & `-promscrape.cluster.memberNum` |

透過 **正確設定 `vmagent` 的 HA 配置**，可以確保 **數據不重複、系統穩定運行、避免數據丟失！** 🚀

