



# 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



Powered by [ChatGPT Exporter](#)



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
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
安裝難度	簡單，單一二進制文件運行	較複雜，需要配置 multiple components (vmstorage, vminsert, vmselect)
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	完整支援	完整支援，並提供額外擴展功能



特色	Prometheus	VictoriaMetrics Cluster
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, vminsert, 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



以繁體中文詳細說明:使用第三方監控工具Prometheus + Grafana



### 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="victoriametrics"}[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



## 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





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





KPI	Prometheus 查詢語法	說明
HTTP 錯誤率	`rate(http_requests_total{status=~"5.."}[5m]) / rate(http_requests_total[5m])`	計算 HTTP 5XX 錯誤的比率
請求延遲 (P99)	`histogram_quantile(0.99, rate(http_request_duration_seconds_bucket[5m]))`	監控 P99 延遲時間
請求延遲 (平均值)	`rate(http_request_duration_seconds_sum[5m]) / rate(http_request_duration_seconds_count[5m])`	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
<b>CPU 使用率</b>	監控各系統的 CPU 消耗	`docker stats` 或 `process_cpu_seconds_total`
<b>記憶體 使用量</b>	監控記憶體消耗	`docker stats` 或 `process_resident_memory_bytes`
<b>I/O 負載</b>	監測寫入/讀取速率	`node_disk_io_time_seconds_total`
<b>數據存儲壓縮率</b>	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)
<b>數據查詢延遲</b>	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]))`
<b>數據寫入吞吐量</b>	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)
<b>HTTP API 韻應時間</b>	測試 `/api/v1/query_range` API 的延遲	使用 `curl` 或 `wrk` 工具進行測試
<b>遠端存儲延遲</b>	比較兩者 `remote_write` 和 `remote_read` 的速度	`rate(remote_write_requests_total[5m])`

## 2. 測試環境統一

為了讓測試環境公平，需要確保：

1. 相同的硬體環境 (CPU 核心數、記憶體大小、磁碟 IOPS)





2. 相同的測試數據來源（相同的 metrics 和寫入速率）
  3. 相同的監控工具（使用 `Prometheus` 和 `Grafana` 來監控 VictoriaMetrics 和 Prometheus 自身）
  4. 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:  
  vmsstorage-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
<b>CPU 核心數 (vCPU)</b>	確保 VictoriaMetrics 和 Prometheus 運行在相同的 CPU 配置上	`lscpu` / `cat /proc/cpuinfo`
<b>記憶體大小</b>	確保兩者可用的記憶體相等	`free -h` / `docker stats`
<b>磁碟 IOPS (Input/Output Per Second)</b>	確保兩者使用相同的磁碟 IOPS	`fio` 測試
<b>網絡帶寬</b>	確保兩者的網絡流量受相同限制	`iperf3` 測試
<b>Docker 資源限制</b>	限制容器 CPU / Memory，避免資源佔用不均	`docker run --cpus` / `docker run --memory`





參數	說明	測試方式
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`





項目	設定方式
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）
<b>vmstorage</b>	負責存儲所有數據	2GB	4GB+
<b>vminsert</b>	負責數據寫入	512MB	1GB+
<b>vmselect</b>	負責查詢請求	1GB	2GB+
<b>vmagent</b>	收集 metrics 並發送到 vminsert	256MB	512MB+
<b>vmalert</b>	負責告警（Alerting）	128MB	256MB+
<b>總計</b>	-	<b>約 4GB</b>	<b>8GB 以上</b>

“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 純 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="victoriametrics"}
```

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
<b>vmstorage</b>	5GB	3GB
<b>vmselect</b>	3GB	2GB
<b>vminsert</b>	2GB	1GB
<b>vmagent</b>	1GB	512MB
<b>vmalert</b>	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



Powered by [ChatGPT Exporter](#)



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。

[VICTORIAMETRICS.COM.CN](http://VICTORIAMETRICS.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](http://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](http://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
```



Powered by [ChatGPT Exporter](#)



```
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)
<b>vmstorage</b>	存儲所有時序數據	1 核	2-4 核
<b>vminsert</b>	負責數據寫入	0.5 核	1 核
<b>vmselect</b>	處理查詢請求	1 核	2-4 核
<b>vmagent</b>	收集 metrics 並發送到 vminsert	0.5 核	1 核
<b>vmalert</b>	負責告警	0.25 核	0.5 核
<b>總計</b>	-	<b>3.25 核</b>	<b>7-8 核</b>

“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 核心

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** 紿 **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 紿 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="victoriamerics"}[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
<b>vmstorage</b>	3 核	1.5 核
<b>vmselect</b>	2 核	1 核
<b>vminsert</b>	1 核	0.5 核
<b>vmagent</b>	1 核	0.5 核
<b>vmalert</b>	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：



Powered by [ChatGPT Exporter](#)



- **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`)：

```
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
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])`	計算一天內存儲的數據筆數
磁碟使用量	`prometheus_tsdb_storage_blocks_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(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、`promtheus`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：



Powered by [ChatGPT Exporter](#)



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://vmstorage: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

1. 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 聽聽端口
  - `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`
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:  

```

### 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
`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` 啟動後崩潰	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！ 

