



# Prometheus vs VictoriaMetrics



詳細說明怎麼比較promethe、victoriametric-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
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, vmaalert 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、vmaalert）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、vmaalert) 本身也能暴露 Prometheus 指標，可在 `prometheus.yml` 中新增：

```
- job_name: 'victoriametrics'
  static_configs:
    - targets: ['vmstorage:8428', 'vminsert:8480', 'vmselect:8481',
'vmagent:8429', 'vmaalert: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	<code>`rate(process_cpu_seconds_total[5m])`</code>	計算 5 分鐘內的 CPU 使用率
記憶體使用量	<code>`process_resident_memory_bytes`</code>	監控應用的實際記憶體使用量
記憶體使用百分比	<code>`(process_resident_memory_bytes / process_virtual_memory_bytes) * 100`</code>	監控記憶體使用比例
磁碟 I/O	<code>`rate(node_disk_io_time_seconds_total[5m])`</code>	監測磁碟讀寫操作時間
磁碟空間剩餘量	<code>`node_filesystem_free_bytes`</code>	監控剩餘磁碟空間
磁碟使用率	<code>`(node_filesystem_size_bytes - node_filesystem_free_bytes) / node_filesystem_size_bytes * 100`</code>	計算磁碟使用百分比





KPI	Prometheus 查詢語法	說明
網絡流量	<code>`rate(node_network_receive_bytes_total[5m])`</code>	監控網絡接收速率
網絡吞吐量	<code>`rate(node_network_transmit_bytes_total[5m])`</code>	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	<code>`rate(container_cpu_usage_seconds_total[5m])`</code>	監控 5 分鐘內容器的 CPU 使用
容器記憶體使用量	<code>`container_memory_usage_bytes`</code>	監測容器當前使用的記憶體
容器磁碟讀取量	<code>`rate(container_fs_reads_bytes_total[5m])`</code>	監控容器的磁碟讀取流量
容器磁碟寫入量	<code>`rate(container_fs_writes_bytes_total[5m])`</code>	監控容器的磁碟寫入流量
容器重啟次數	<code>`increase(kube_pod_container_status_restarts_total[5m])`</code>	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)	<code>`vm_tsdb_time_series`</code>	監測當前存儲的時序數據量
數據寫入速率	<code>`rate(vm_inserted_rows_total[5m])`</code>	監測 5 分鐘內寫入 VictoriaMetrics 的數據行數
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。

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





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

這些 KPI 可搭配 **vmalet** 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, vmaalert 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、vmaalert）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	<code>`docker stats`</code> 或 <code>`process_cpu_seconds_total`</code>
<b>記憶體使用量</b>	監控記憶體消耗	<code>`docker stats`</code> 或 <code>`process_resident_memory_bytes`</code>
<b>磁碟 I/O 負載</b>	監測寫入/讀取速率	<code>`node_disk_io_time_seconds_total`</code>
<b>數據存儲壓縮率</b>	比較存儲相同數據所需的磁碟空間	<code>`du -sh /var/lib/prometheus`</code> (Prometheus) vs. <code>`du -sh /storage`</code> (VictoriaMetrics)
<b>數據查詢延遲</b>	比較 PromQL 查詢相同指標時的響應時間	<code>`histogram_quantile(0.99, rate(http_request_duration_seconds_bucket[5m]))`</code>
<b>數據寫入吞吐量</b>	測試每秒可處理的 metrics 條數	<code>`rate(vm_inserted_rows_total[5m])`</code> (VM) vs. <code>`rate(prometheus_tsdb_head_samples_appended_total[5m])`</code> (Prometheus)
<b>HTTP API 響應時間</b>	測試 <code>`/api/v1/query_range`</code> API 的延遲	使用 <code>`curl`</code> 或 <code>`wrk`</code> 工具進行測試
<b>遠端存儲延遲</b>	比較兩者 <code>`remote_write`</code> 和 <code>`remote_read`</code> 的速度	<code>`rate(remote_write_requests_total[5m])`</code>

## 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:
  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	<code>`lscpu`</code> / <code>`cat /proc/cpuinfo`</code>
記憶體大小	確保兩者可用的記憶體相等	<code>`free -h`</code> / <code>`docker stats`</code>
磁碟 IOPS（Input/Output Per Second）	確保兩者使用相同的磁碟 IOPS	<code>`fio`</code> 測試
網絡帶寬	確保兩者的網絡流量受相同限制	<code>`iperf3`</code> 測試
Docker 資源限制	限制容器 CPU / Memory，避免資源佔用不均	<code>`docker run --cpus`</code> / <code>`docker run --memory`</code>



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 tbf 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	<code>`iperf3`</code> 測試 + <code>`tc qdisc`</code> 限制
測試數據	<code>`vmagent`</code> 或 <code>`stress-ng`</code> 產生相同數據
測試時間	<code>`timeout 3600s`</code> 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！🚀