



VictoriaMetrics Docker Cluster



Cluster version.pdf

PDF

參考此pdf，以繁體中文詳細說明用docker image 而不使用k8s，建立cluster VictoriaMetrics



要在不使用 Kubernetes 的情況下，利用 Docker 建立 VictoriaMetrics Cluster，請依照以下步驟操作：

1. 確認系統環境

- 安裝 Docker 並確保其服務正常運作。
- 為每個 VictoriaMetrics 服務分配適當的主機或虛擬機 (``vminsert``、``vmselect`` 和 `vmstorage`），以實現服務解耦和水平擴展。

2. 拉取 Docker 映像檔

拉取 VictoriaMetrics Cluster 所需的 Docker 映像檔：

```
docker pull victoriametrics/vminsert:<version>
docker pull victoriametrics/vmselect:<version>
docker pull victoriametrics/vmstorage:<version>
```

請將 `<version>` 替換為所需版本。



Powered by [ChatGPT Exporter](#)



3. 部署 `vmstorage`

啟動用於儲存資料的 `vmstorage` 節點：

```
docker run -d \
--name vmstorage \
-p 8482:8482 \
-v /path/to/storage:/storage \
victoriametrics/vmstorage:<version> \
-retentionPeriod=<days> \
-storageDataPath=/storage
```

- **`-retentionPeriod`**：資料保留天數。
- **`/path/to/storage`**：本地資料目錄。

4. 部署 `vminsert`

啟動用於接收數據並分散到 `vmstorage` 節點的 `vminsert` 節點：

```
docker run -d \
--name vminsert \
-p 8480:8480 \
victoriametrics/vminsert:<version> \
-storageNode=<vmstorage_host>:8482
```

- **`-storageNode`**：指向 `vmstorage` 節點的地址。

5. 部署 `vmselect`

啟動用於查詢的 `vmselect` 節點：





```
docker run -d \
--name vmselect \
-p 8481:8481 \
victoriametrics/vmselect:<version> \
-storageNode=<vmstorage_host>:8482
```

- `-storageNode`：指向 `vmstorage` 節點的地址。

6. 配置負載均衡（可選）

如果部署多個 `vminsert` 或 `vmselect` 節點，可使用 Nginx 作為負載均衡器：

Nginx 配置範例：

```
http {
    upstream vminsert {
        server <vminsert1>:8480;
        server <vminsert2>:8480;
    }

    upstream vmselect {
        server <vmselect1>:8481;
        server <vmselect2>:8481;
    }

    server {
        location /insert/ {
            proxy_pass http://vminsert;
        }
        location /select/ {
            proxy_pass http://vmselect;
        }
    }
}
```





7. 測試服務

- 驗證數據寫入：

```
curl -X POST "http://<vminsert_host>:8480/insert/0/prometheus/api/v1/write"
 \
 -d '<data_payload>'
```

- 驗證查詢：

```
curl "http://<vmselect_host>:8481/select/0/prometheus/api/v1/query?
query=up"
```

8. 持續監控與擴展

- 定期監控服務的性能與資源使用情況。
- 當負載增加時，可以通過增加節點數量來水平擴展 `vminsert`、`vmselect` 和 `vmstorage`。

如果需要更詳細的技術支持或設定，建議參考 [VictoriaMetrics 官方文檔](#)。



以下是參考《Cluster version.pdf》，以繁體中文詳細說明如何使用 **vmagent** 測試 VictoriaMetrics Cluster 的方法。

參考"Cluster version.pdf"，以繁體中文詳細說明怎麼跟
vmagent測試





1. 安裝與配置 vmagent

vmagent 是一個高效的數據收集代理，用於從多個來源收集監控數據並將其發送到 VictoriaMetrics。

(1) 拉取 vmagent Docker 映像檔

```
docker pull victoriametrics/vmagent:<version>
```

將 `<version>` 替換為所需的版本。

(2) 配置 vmagent

建立 **vmagent** 的配置文件，例如 `vmagent.yml`，定義監控目標 (targets)：

```
global:
  scrape_interval: 15s
  scrape_timeout: 10s

scrape_configs:
  - job_name: 'example_job'
    static_configs:
      - targets:
          - '<target_host>:<port>'
```

- `scrape_interval`：數據收集的間隔時間，預設為 15 秒。
- `targets`：目標服務的地址，例如 `localhost:9100`。

(3) 啟動 vmagent

使用以下命令啟動 **vmagent**，並配置將數據發送到 VictoriaMetrics Cluster 的 `vminsert` 節點：

```
docker run -d \
  --name vmagent \
  -p 8429:8429 \
  -v /path/to/vmagent.yml:/etc/vmagent.yml \
  victoriametrics/vmagent:<version> \
```





```
-remoteWrite.url=http://<vminsert_host>:8480/insert/0/prometheus/api/v1/write
 \
 -promscrape.config=/etc/vmagent.yml
```

- `remoteWrite.url`：設置數據寫入的目標 URL，指向 `vminsert` 節點。
- `promscrape.config`：設置 Prometheus 格式的配置文件。

2. 測試數據寫入

(1) 模擬數據寫入

vmagent 會自動從配置的目標 (targets) 中抓取數據，並將其寫入 VictoriaMetrics Cluster 的 `vminsert` 節點。

可以在目標服務上安裝 [Node Exporter](#) 或其他兼容的數據來源，然後在配置中添加目標地址：

```
static_configs:
  - targets:
    - 'localhost:9100'
```

(2) 驗證數據是否成功寫入

執行以下查詢，驗證數據是否已寫入 `vmselect` 節點：

```
curl "http://<vmselect_host>:8481/select/0/prometheus/api/v1/query?query=up"
```

3. 監控 vmagent 本身的運行狀態

vmagent 提供 `/metrics` 頁面，用於暴露其自身的運行指標。使用以下 URL 獲取 **vmagent** 的監控數據：



```
curl http://<vmagent_host>:8429/metrics
```

這些數據可用於監控 **vmagent** 的效能、資源使用及數據抓取的狀態。

4. 啟用多目標數據寫入

如果需要將數據寫入多個 VictoriaMetrics Cluster 節點，可以在啟動參數中添加多個 `--remoteWrite.url`：

```
-remoteWrite.url=http://<vminsert1>:8480/insert/0/prometheus/api/v1/write \
-remoteWrite.url=http://<vminsert2>:8480/insert/0/prometheus/api/v1/write
```

5. 常見測試與排錯方法

(1) 檢查 **vmagent** 配置

確保 `vmagent.yaml` 中的目標地址正確，且所有目標服務均正常運行。

(2) 驗證數據抓取狀態

訪問以下 URL 檢查 **vmagent** 的當前抓取目標：

```
curl http://<vmagent_host>:8429/targets
```

該頁面會顯示所有目標的抓取狀態及錯誤信息。

(3) 驗證數據傳輸

查看 **vminsert** 的 `/metrics` 頁面，確認數據是否成功傳輸：

```
curl http://<vminsert_host>:8480/metrics
```





(4) 測試查詢數據

使用 `vmselect` 執行以下查詢以檢查數據：

```
curl "http://<vmselect_host>:8481/select/0/prometheus/api/v1/query_range?  
query=up&start=<start_time>&end=<end_time>&step=15s"
```

總結

利用 **vmagent** 測試 VictoriaMetrics Cluster 時，需確認以下幾個環節：

1. **vmagent** 配置正確，並成功抓取目標服務的數據。
2. **vminsert** 節點能正確接收並存儲數據。
3. **vmselect** 節點能正確查詢數據。

通過這些步驟，可以確保 **vmagent** 與 VictoriaMetrics Cluster 的整合和測試順利完成。



參考"Cluster version.pdf"，以繁體中文詳細說明怎麼跟
grafana測試



以下是參考《Cluster version.pdf》所提供的資訊，詳細說明如何將 VictoriaMetrics Cluster 與 **Grafana** 整合並進行測試。

1. 安裝與啟動 Grafana

(1) 使用 Docker 安裝 Grafana

執行以下命令拉取並啟動 Grafana：



Powered by [ChatGPT Exporter](#)



```
docker pull grafana/grafana:<version>
docker run -d \
  --name=grafana \
  -p 3000:3000 \
  grafana/grafana:<version>
```

將 `<version>` 替換為需要的 Grafana 版本。

(2) 登入 Grafana

- 預設 Grafana 網頁界面地址為 `http://<grafana_host>:3000`。
- 登入預設帳號：
 - 使用者名稱：`admin`
 - 密碼：`admin`
- 第一次登入時系統會要求修改密碼。

2. 配置 VictoriaMetrics Cluster 資料來源

(1) 在 Grafana 中新增資料來源

1. 進入 Grafana 網頁界面。
2. 點擊左側邊欄的齒輪圖標 (Settings)。
3. 選擇 **Data Sources**，然後點擊 **Add data source**。
4. 在列表中選擇 **Prometheus** 作為資料來源類型。

(2) 配置資料來源

填寫以下字段：

- **URL**：
 - 格式為 `http://<vmselect_host>:8481/select/0/prometheus/`
 - 其中 `<vmselect_host>` 為 `vmselect` 節點的主機地址。





- **Access**：選擇 `Server (default)`。
 - 點擊 **Save & Test**，確認資料來源配置正確。
-

3. 匯入官方 Dashboard

VictoriaMetrics 提供了官方的 Grafana Dashboard，可用於監控 Cluster 的運行狀態。

(1) 下載 Dashboard JSON

前往 [Grafana 官方 Dashboard](#) 或 VictoriaMetrics 提供的資源頁面下載 JSON 檔案（如 Dashboard ID `11176`）。

(2) 在 Grafana 中匯入 Dashboard

1. 點擊左側邊欄的 + (Create)。
 2. 選擇 **Import**。
 3. 選擇以下方式之一匯入：
 - **匯入 ID**：輸入 `11176`，然後點擊 **Load**。
 - **匯入 JSON**：將下載的 JSON 文件內容貼入文本框。
 4. 選擇剛剛配置的 VictoriaMetrics 資料來源，然後點擊 **Import**。
-

4. 測試數據查詢

(1) 添加圖表進行測試

1. 點擊左側邊欄的 + (Create)，選擇 **Dashboard**。
2. 點擊 **Add new panel**。
3. 在查詢區域選擇資料來源為剛配置的 VictoriaMetrics。
4. 輸入以下 PromQL 查詢進行測試：





up

此查詢會返回所有服務的健康狀態。

5. 配置圖表樣式後點擊 **Apply** 保存。

(2) 測試查詢範例

- 查看系統的監控指標，例如：

```
vm_http_requests_total
```

- 檢查 VM 集群中每秒處理的請求量：

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

5. 常見問題排查

(1) 資料來源無法連線

- 確認 Grafana 與 `vmselect` 節點在同一網段或允許跨網段連接。
- 檢查 `vmselect` 的 8481 埠是否開放。

(2) 查詢無法返回數據

- 確認 `vminsert` 是否已成功寫入數據至 `vmstorage`。
- 確保 PromQL 查詢語法正確。

(3) 性能問題

- 使用 Grafana 官方 Dashboard 檢查集群資源使用情況，例如 CPU、記憶體或磁碟的佔用率。

總結





通過上述步驟，您可以將 VictoriaMetrics Cluster 與 Grafana 整合，並利用 Grafana 監控數據指標與系統性能。這樣的設置有助於實時了解集群運行狀況，並快速診斷問題。



參考"Cluster version.pdf"，以繁體中文詳細說明若**vmagent** 失效



根據《Cluster version.pdf》內容，以下是針對 **vmagent** 失效時的詳細說明及處理方式：

1. **vmagent** 失效的可能影響

vmagent 是負責從多個監控目標收集數據並將其寫入 VictoriaMetrics 的代理工具。如果 **vmagent** 發生失效，可能會造成以下影響：

- **數據收集中斷**：來自 Prometheus、Node Exporter 或其他監控目標的數據無法傳輸至 VictoriaMetrics。
- **數據缺失**：期間產生的監控數據無法存儲，導致查詢和可視化時顯示不完整。
- **寫入壓力轉移**：如果有多个 **vmagent** 節點，其餘節點可能會面臨更大的寫入壓力。

2. 檢測 **vmagent** 失效的症狀

以下是一些常見的症狀，可以幫助判斷 **vmagent** 是否失效：

1. 無法抓取目標數據：

- 在 **vmagent** 的 `/targets` 頁面無法查看到任何目標數據。

```
curl http://<vmagent_host>:8429/targets
```





2. 無法寫入數據至 **vminsert** :

- 查看 **vminsert** 節點的 `/metrics`，檢查是否有數據接收減少或中斷。

```
curl http://<vminsert_host>:8480/metrics
```

3. **Grafana** 中無法查詢到最新數據 :

- 在 **Grafana** 上執行 PromQL 查詢時，數據顯示為空或僅有歷史數據。

3. **vmagent** 失效的可能原因

1. 配置錯誤 :

- 配置檔案 `vmagent.yml` 中的目標地址 (targets) 或寫入地址 (remoteWrite.url) 設定錯誤。

2. 資源不足 :

- CPU、記憶體或磁碟空間不足，導致 **vmagent** 無法正常運行。

3. 網路問題 :

- **vmagent** 無法連接到目標或 `vminsert` 節點，可能是因網路配置錯誤。

4. 程序崩潰 :

- **vmagent** 進程意外終止。

4. 處理 **vmagent** 失效的步驟

(1) 檢查 **vmagent** 服務狀態

使用以下命令檢查 **vmagent** 的容器或進程是否運行：

```
docker ps | grep vmagent
```





若容器未運行，可以使用以下命令重新啟動：

```
docker restart vmagent
```

(2) 查看 **vmagent** 日誌

檢查 **vmagent** 的運行日誌以診斷問題：

```
docker logs vmagent
```

關鍵日誌信息包括：

- **target scraping errors** (目標抓取錯誤)。
- **remoteWrite errors** (數據寫入錯誤)。

(3) 檢查配置檔案

驗證 `vmagent.yml` 配置是否正確，特別是以下部分：

- **targets**：目標地址是否可訪問。
- **remoteWrite.url**：是否正確指向 `vminsert` 節點。

示例：

```
scrape_configs:  
  - job_name: 'example_job'  
    static_configs:  
      - targets:  
        - '<valid_target_host>:<port>'
```

(4) 確認資源是否充足

檢查主機的資源狀態 (CPU、記憶體、磁碟空間)：

```
top # 查看 CPU 和記憶體使用率  
df -h # 查看磁碟空間
```





如果資源不足，可以通過升級硬體或分散負載來解決。

(5) 測試連線

確保 **vmagent** 能連接到目標和 `vminsert` 節點：

```
curl http://<target_host>:<port>/metrics # 測試目標連線  
curl http://<vminsert_host>:8480/metrics # 測試 vminsert 連線
```

(6) 啟用監控與告警

為 **vmagent** 設置監控和告警，以便及時發現問題。例如：

- 在 Grafana 中監控 **vmagent** 的指標 (`vmagent_scrape_samples_scraped`)。
- 設置告警條件，如數據抓取速率低於預期。

5. 高可用性配置建議

(1) 部署多個 **vmagent** 節點

為提高可用性，建議部署多個 **vmagent** 節點，並在負載均衡器（如 Nginx）中配置輪詢。

(2) 配置多目標寫入

在 **vmagent** 中配置多個 `-remoteWrite.url`，以確保數據可同時寫入多個 `vminsert` 節點：

```
-remoteWrite.url=http://<vminsert1>:8480/insert/0/prometheus/api/v1/write \  
-remoteWrite.url=http://<vminsert2>:8480/insert/0/prometheus/api/v1/write
```

(3) 啟用自動重啟

為 **vmagent** 配置 Docker 的自動重啟策略：

```
docker update --restart always vmagent
```





總結

若 **vmagent** 發生失效，需快速檢查其運行狀態、配置檔案和系統資源，並採取修復措施。同時，透過高可用性配置（多節點部署、多目標寫入），可有效降低失效的影響並提升系統穩定性。



參考"Cluster version.pdf"，以繁體中文詳細說明若grafana
失效



當 **Grafana** 發生失效時，會影響監控數據的可視化與查詢。以下是根據《Cluster version.pdf》的資料，詳細說明 **Grafana** 失效時的影響、原因分析及處理方法。

1. Grafana 失效的可能影響

1. 無法可視化數據：

- 使用者無法透過圖形界面檢視 VictoriaMetrics Cluster 的監控數據。

2. 監控與告警中斷：

- 設置在 Grafana 中的告警規則無法觸發。

3. 查詢受阻：

- 通過 Grafana 執行的 PromQL 查詢將無法運作。

2. Grafana 失效的可能原因





1. Grafana 容器或服務停止：

- 容器未啟動或服務崩潰。

2. 資源不足：

- 主機的 CPU、記憶體或磁碟空間不足，導致 Grafana 無法正常運行。

3. 網路問題：

- Grafana 與 VictoriaMetrics 的 `vmselect` 節點無法連線。

4. 配置錯誤：

- 資料來源 (Data Source) 配置不正確或丟失。

5. 插件問題：

- Grafana 中的插件更新或錯誤導致功能無法正常使用。

3. 檢測 Grafana 失效的症狀

(1) Grafana 網頁界面無法訪問

在瀏覽器中打開 `http://<grafana_host>:3000`，如果界面無法加載，可能是 Grafana 容器未運行或網路問題。

(2) 資料來源無法連線

- 在 Grafana 的 **Data Sources** 頁面中測試資料來源，顯示連線失敗。

(3) 查詢報錯或無法返回數據

- 在面板中執行 PromQL 查詢時出現錯誤，或查詢結果為空。

(4) 無法啟動告警

- 設置的告警規則未觸發或報錯。





4. Grafana 失效的處理步驟

(1) 確認 Grafana 服務狀態

檢查 Docker 容器或系統服務是否正常運行：

```
docker ps | grep grafana
```

若容器未啟動，使用以下命令重啟：

```
docker restart grafana
```

若使用系統服務管理 Grafana：

```
sudo systemctl status grafana-server  
sudo systemctl restart grafana-server
```

(2) 查看 Grafana 日誌

檢查 Grafana 的日誌以診斷問題：

```
docker logs grafana
```

或：

```
sudo journalctl -u grafana-server
```

關鍵日誌信息包括：

- **資料來源錯誤**（例如：無法連接到 `vmselect`）。
- **資源限制**（如記憶體不足導致崩潰）。
- **插件錯誤**（如插件載入失敗）。

(3) 測試資料來源連線





1. 在 Grafana 的 **Data Sources** 頁面中，選擇已配置的資料來源並點擊 **Save & Test**。

2. 如果連線失敗：

- 確認 `vmselect` 節點是否可用：

```
curl http://<vmselect_host>:8481/metrics
```

- 檢查資料來源 URL 是否正確，例如：

```
http://<vmselect_host>:8481/select/0/prometheus/
```

(4) 檢查系統資源

確認主機的資源狀態：

```
top      # 檢查 CPU 和記憶體  
df -h    # 檢查磁碟空間
```

若資源不足，考慮升級硬體或清理不必要的檔案。

(5) 修復配置問題

若資料來源或面板配置錯誤：

1. 重新配置資料來源，確保指向正確的 `vmselect` 節點。

2. 在面板編輯頁面中，檢查並修正 PromQL 查詢語法。

(6) 測試告警功能

在 **Alerting** 頁面中測試告警是否正常觸發：

1. 查看告警規則狀態。

2. 如果告警無法觸發，檢查 PromQL 查詢是否正確。

(7) 更新或修復插件

如果問題與插件相關：

1. 在 **Plugins** 頁面中檢查插件狀態。



2. 移除或更新出現問題的插件。

5. 預防與高可用性配置建議

(1) 啟用自動重啟

為 Grafana 容器啟用自動重啟策略：

```
docker update --restart always grafana
```

(2) 設置多節點 Grafana

部署多個 Grafana 節點以實現高可用性，並使用負載均衡器（如 Nginx）進行流量分配。

(3) 定期備份 Grafana 配置

使用 Grafana 的匯出功能，定期備份以下內容：

- 資料來源配置。
- Dashboard 配置。
- 告警規則。

(4) 配置資源監控

在監控系統中加入 Grafana 的資源指標，定期檢查 Grafana 的 CPU、記憶體和磁碟使用情況。

6. 排錯範例：實際操作步驟

假設 Grafana 的面板查詢報錯，執行以下步驟：

1. 確認 **vmselect** 是否可用：

```
curl http://<vmselect_host>:8481/select/0/prometheus/api/v1/query?query=up
```





2. 測試 Grafana 資料來源：

- 在 **Data Sources** 中選擇資料來源，點擊 **Save & Test**。

3. 修改 PromQL 查詢：

- 確保查詢格式正確，例如：

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

總結

當 Grafana 失效時，需快速檢查服務狀態、資料來源配置及系統資源，並採取對應的修復措施。通過高可用性配置和定期備份，可有效降低 Grafana 失效對監控系統的影響。



參考"Cluster version.pdf"，以繁體中文詳細說明若vminsert
失效



當 **vminsert** 節點失效時，VictoriaMetrics Cluster 的數據寫入功能將受到影響。以下根據《Cluster version.pdf》的內容，詳細說明 **vminsert** 失效時的影響、原因分析及處理方法。

1. vminsert 失效的可能影響

vminsert 是負責接收數據並將其分發到 **vmstorage** 節點的服務。若 **vminsert** 失效，可能會導致：

1. 數據寫入中斷：

- 無法接收來自數據來源（如 Prometheus、vmagent）的數據。





2. 數據丟失：

- 在失效期間，無法保存數據。

3. 集群壓力轉移：

- 如果存在多個 **vminsert** 節點，其他節點的負載可能會增加。

2. **vminsert** 失效的可能原因

1. 資源不足：

- CPU、記憶體或磁碟空間不足，導致進程崩潰。

2. 網路問題：

- 無法與數據來源（如 **vmagent**）或 **vmstorage** 節點建立連線。

3. 配置錯誤：

- vminsert** 的啟動參數錯誤，例如指向錯誤的 **vmstorage** 地址。

4. 服務崩潰：

- 程序因未處理的異常而停止運行。

5. 磁碟空間不足：

- 用於緩存的磁碟空間耗盡。

3. 檢測 **vminsert** 失效的症狀

(1) 無法接收數據

- 檢查數據來源（如 **vmagent**）的日誌中是否出現連線錯誤。

(2) 無法存儲數據

- 查看 **vmstorage** 節點的 `metrics` 頁面，檢查是否未收到新的數據。





```
curl http://<vmstorage_host>:8482/metrics
```

(3) 查詢無法返回最新數據

- 在 **vmselect** 執行查詢時，僅能獲取歷史數據。

(4) vminsert 的健康檢查失敗

- 嘗試訪問 **vminsert** 節點的 `/health` 頁面確認狀態：

```
curl http://<vminsert_host>:8480/health
```

4. 處理 vminsert 失效的步驟

(1) 檢查服務狀態

確認 **vminsert** 的容器或進程是否正在運行：

```
docker ps | grep vminsert
```

若容器未運行，使用以下命令啟動：

```
docker restart vminsert
```

若使用系統服務管理：

```
sudo systemctl status vminsert
sudo systemctl restart vminsert
```

(2) 查看日誌以診斷問題

檢查 **vminsert** 的運行日誌：





```
docker logs vminsert
```

或：

```
sudo journalctl -u vminsert
```

關注以下錯誤信息：

- **與 `vmstorage` 連接錯誤。**
- **數據來源連接失敗。**
- **資源不足錯誤** (如 OOM 或磁碟空間不足)。

(3) 驗證配置

檢查 `vminsert` 的啟動參數是否正確：

- **-storageNode** 是否正確指向 `vmstorage` 節點。
- 例如：

```
docker run -d \
--name vminsert \
-p 8480:8480 \
victoriametrics/vminsert:<version> \
-storageNode=<vmstorage_host>:8482
```

(4) 測試數據寫入功能

從數據來源模擬寫入請求：

```
curl -X POST "http://<vminsert_host>:8480/insert/0/prometheus/api/v1/write" \
-d '<data_payload>'
```

- 若返回錯誤，需進一步檢查配置與網路。

(5) 確保資源充足





檢查主機資源：

```
top # 檢查 CPU 和記憶體使用率  
df -h # 檢查磁碟空間
```

若資源不足，考慮升級硬體或分散負載。

(6) 測試 **vmstorage** 節點

確認 **vmstorage** 是否正常運行，並能接收來自 **vminsert** 的數據：

```
curl http://<vmstorage_host>:8482/metrics
```

5. 預防與高可用性配置建議

(1) 部署多個 **vminsert** 節點

為提高可用性，部署多個 **vminsert** 節點，並透過負載均衡器（如 Nginx）進行流量分配。

(2) 配置多目標寫入

在數據來源（如 **vmagent**）中配置多個寫入地址：

```
-remoteWrite.url=http://<vminsert1>:8480/insert/0/prometheus/api/v1/write \
-remoteWrite.url=http://<vminsert2>:8480/insert/0/prometheus/api/v1/write
```

(3) 啟用自動重啟

為 **vminsert** 容器設置自動重啟策略：

```
docker update --restart always vminsert
```

(4) 監控 **vminsert**

使用 Prometheus 或 Grafana 監控 **vminsert** 的健康狀態與性能指標，例如：



- **vm_http_requests_total**：請求總數。
 - **vm_insert_requests_total**：數據寫入請求總數。
-

6. 排錯範例：實際操作步驟

假設 **vminsert** 無法接收數據，執行以下步驟：

1. 確認 **vminsert** 是否運行：

```
docker ps | grep vminsert
```

2. 測試與 **vmstorage** 節點的連線：

```
curl http://<vmstorage_host>:8482/metrics
```

3. 查看 **vminsert** 日誌以診斷問題：

```
docker logs vminsert
```

4. 驗證數據來源是否正確指向 **vminsert**：

- 檢查數據來源的配置檔案。
-

總結

當 **vminsert** 失效時，需快速檢查服務狀態、配置及資源，並透過高可用性配置減少影響。此外，定期監控 **vminsert** 的性能和健康狀況，能有效預防問題的發生。



參考"Cluster version.pdf"，以繁體中文詳細說明若2個
vminsert中的一個失效





當 **VictoriaMetrics Cluster** 中的兩個 **vminsert** 節點之一失效時，集群的數據寫入仍能繼續，但可能導致部分性能下降或負載轉移。以下根據《Cluster version.pdf》的內容，詳細說明 **vminsert** 節點失效的影響、診斷方法及處理方式。

1. 部分 **vminsert** 節點失效的影響

1. 數據寫入性能下降：

- 剩餘的 **vminsert** 節點需處理所有數據寫入請求，可能導致負載增加。

2. 高可用性受限：

- 如果剩餘節點也失效，整個數據寫入功能將中斷。

3. 數據重新路由：

- 數據來源（如 **vmagent** 或 Prometheus）會將請求重新路由到可用的 **vminsert** 節點。

2. 診斷部分 **vminsert** 節點失效的方法

(1) 檢查剩餘節點的負載

- 查看可用 **vminsert** 節點的指標數據，例如：

```
curl http://<active_vminsert_host>:8480/metrics
```

關注以下指標：

- vm_http_requests_total**：請求總數。
- vm_http_request_duration_seconds_sum**：請求的處理時間。

(2) 測試失效的 **vminsert** 節點

- 嘗試訪問失效節點的健康檢查端點：



```
curl http://<failed_vminsert_host>:8480/health
```

如果返回非 `200 OK`，說明該節點確實失效。

(3) 檢查數據來源的行為

- 查看 **vmagent** 或其他數據來源的日誌，確認是否存在重試或連線錯誤。

(4) 查看失效節點的日誌

- 登入失效節點的伺服器，檢查日誌信息以診斷問題：

```
docker logs vminsert
```

關注以下錯誤：

- 與 **vmstorage** 連線失敗。
- 資源不足（如 OOM 或磁碟空間不足）。

3. 部分 vminsert 節點失效的處理步驟

(1) 檢查失效節點的狀態

確認容器或服務是否運行：

```
docker ps | grep vminsert
```

如果未運行，嘗試重啟：

```
docker restart vminsert
```

若服務未啟動，檢查系統服務：

```
sudo systemctl status vminsert
sudo systemctl restart vminsert
```





(2) 確保系統資源充足

- 檢查伺服器的 CPU、記憶體和磁碟空間：

```
top      # 檢查 CPU 和記憶體  
df -h    # 檢查磁碟空間
```

- 如果資源不足，請釋放空間或升級硬體。

(3) 測試與 **vmstorage** 節點的連線

確認失效的 **vminsert** 能連接到 **vmstorage**：

```
curl http://<vmstorage_host>:8482/metrics
```

若無法連接，檢查網路連線和 **vmstorage** 節點的運行狀態。

(4) 修復或替換失效節點

若節點無法恢復，考慮替換節點：

1. 停止失效的節點。
2. 部署一個新的 **vminsert** 節點，指向相同的 **vmstorage** 節點：

```
docker run -d \  
  --name vminsert-new \  
  -p 8480:8480 \  
  victoriametrics/vminsert:<version> \  
  -storageNode=<vmstorage_host>:8482
```

4. 優化配置以減少影響

(1) 配置多目標寫入

在數據來源中配置多個 **vminsert** 節點，實現自動故障轉移。例如，在 **vmagent** 中：



```
remoteWrite:  
  - url: http://<vminsert1_host>:8480/insert/0/prometheus/api/v1/write  
  - url: http://<vminsert2_host>:8480/insert/0/prometheus/api/v1/write
```

(2) 使用負載均衡器

在 **vminsert** 節點前部署負載均衡器（如 Nginx 或 HAProxy），實現自動流量分配。範例 Nginx 配置：

```
upstream vminsert {  
    server <vminsert1_host>:8480;  
    server <vminsert2_host>:8480;  
}  
  
server {  
    listen 8480;  
    location / {  
        proxy_pass http://vminsert;  
    }  
}
```

(3) 啟用自動重啟策略

為所有 **vminsert** 節點配置 Docker 的自動重啟策略：

```
docker update --restart always vminsert
```

5. 排錯範例：實際操作步驟

假設有兩個 **vminsert** 節點，其中一個失效：

1. 確認失效節點的狀態：

```
docker ps | grep vminsert
```





2. 測試失效節點的健康檢查：

```
curl http://<failed_vminsert_host>:8480/health
```

3. 重啟失效節點：

```
docker restart vminsert
```

4. 如果重啟無效，檢查日誌以診斷問題：

```
docker logs vminsert
```

5. 如果問題無法修復，部署新節點替代：

```
docker run -d \
--name vminsert-new \
-p 8480:8480 \
victoriametrics/vminsert:<version> \
--storageNode=<vmstorage_host>:8482
```

6. 結論

當兩個 **vminsert** 節點之一失效時，VictoriaMetrics Cluster 仍能部分運行，但性能和穩定性可能受到影響。通過以下措施可減少影響：

- 部署多個 **vminsert** 節點。
- 配置負載均衡和多目標寫入。
- 定期監控節點健康狀況。

這樣的配置能確保集群在部分節點失效時仍能穩定運行，並快速恢復功能。



參考"Cluster version.pdf"，以繁體中文詳細說明若
vmstorage失效





當 **vmstorage** 節點失效時，會對 **VictoriaMetrics Cluster** 的數據存取產生影響，以下根據《Cluster version.pdf》的內容詳細說明 **vmstorage** 節點失效的影響、診斷方法及處理步驟。

1. **vmstorage** 節點失效的可能影響

1. 數據寫入受影響：

- **vminsert** 無法將數據寫入失效的 **vmstorage** 節點。
- 數據會重新路由到其他健康的 **vmstorage** 節點，導致負載增加。

2. 數據查詢不完整：

- **vmselect** 在查詢數據時可能缺失存儲於失效節點的部分數據。
- 查詢結果可能顯示為 **partial response**。

3. 集群性能下降：

- 健康節點需要承擔更高的負載，可能導致性能下降。

2. 診斷 **vmstorage** 節點失效的方法

(1) 驗證健康檢查端點

測試失效的 **vmstorage** 節點的健康狀態：

```
curl http://<vmstorage_host>:8482/health
```

- 返回 `200 OK` 表示正常。
- 返回其他狀態或無響應則可能失效。

(2) 檢查 **vminsert** 日誌





查看 **vminsert** 是否報告與失效的 **vmstorage** 節點連線錯誤：

```
docker logs vminsert
```

關鍵錯誤信息：

- 無法連接 **vmstorage**。
- **storageNode unreachable**。

(3) 檢查 **vmselect** 查詢行為

在 **vmselect** 中執行查詢，檢查是否有 **partial response**：

```
curl "http://<vmselect_host>:8481/select/0/prometheus/api/v1/query?query=up"
```

- 如果返回部分數據且包含 `{"isPartial": true}`，表示部分 **vmstorage** 節點無法提供數據。

(4) 查看失效節點的日誌

登入失效的 **vmstorage** 節點伺服器，查看運行日誌：

```
docker logs vmstorage
```

關注以下錯誤：

- 磁碟空間不足。
- **OOM (記憶體不足)**。
- 與其他節點的網路連接失敗。

3. **vmstorage** 節點失效的處理步驟

(1) 檢查服務狀態



確認 **vmstorage** 的容器或服務是否正在運行：

```
docker ps | grep vmstorage
```

若未運行，嘗試啟動：

```
docker restart vmstorage
```

若服務未啟動，檢查系統服務：

```
sudo systemctl status vmstorage  
sudo systemctl restart vmstorage
```

(2) 檢查資源使用情況

- 確認伺服器的 CPU、記憶體和磁碟空間是否充足：

```
top      # 檢查 CPU 和記憶體  
df -h    # 檢查磁碟空間
```

- 如果磁碟空間不足，可以刪除無用檔案或擴展存儲。

(3) 測試與其他節點的連線

確認失效節點是否能與其他節點正常通信：

```
ping <other_node_host>  
curl http://<other_vmstorage_host>:8482/metrics
```

(4) 重啟或替換失效節點

如果無法修復失效節點，可以部署新的 **vmstorage** 節點：

- 停止失效的節點。
- 部署新的 **vmstorage** 節點：





```
docker run -d \
--name vmstorage-new \
-p 8482:8482 \
-v /path/to/storage:/storage \
victoriametrics/vmstorage:<version> \
-retentionPeriod=<days> \
-storageDataPath=/storage
```

3. 更新 **vminsert** 和 **vmselect** 的配置以包含新節點：

- **vminsert** :

```
docker restart vminsert --storageNode=<new_vmstorage_host>:8482
```

- **vmselect** :

```
docker restart vmselect --storageNode=<new_vmstorage_host>:8482
```

(5) 測試集群功能

- 測試數據寫入：

```
curl -X POST "http://<vminsert_host>:8480/insert/0/prometheus/api/v1/write"
\
-d '<data_payload>'
```

- 測試數據查詢：

```
curl "http://<vmselect_host>:8481/select/0/prometheus/api/v1/query?
query=up"
```

4. 優化與高可用性配置建議

(1) 部署多個 **vmstorage** 節點

增加 **vmstorage** 節點數量，分散數據存儲壓力，提高集群穩定性。





(2) 啟用數據複製

為了避免單節點故障導致數據丟失，可以啟用數據複製功能：

- 配置 **vminsert** 的 `replicationFactor=N`，設定複製數量。
- 複製數據將存儲於多個 **vmstorage** 節點。

(3) 使用自動發現功能

使用文件或 DNS SRV 的方式，自動發現並更新 **vmstorage** 節點：

- 文件方式：

```
-storageNode=file:/path/to/vmstorage-list
```

- DNS SRV：

```
-storageNode=srv+vmstorage-autodiscovery
```

(4) 定期監控與告警

設置監控和告警規則，監控以下指標：

- vm_rpc_vmstorage_is_reachable**：監控節點是否可達。
- vm_storage_is_read_only**：監控節點是否進入只讀模式。

5. 排錯範例：實際操作步驟

假設有三個 **vmstorage** 節點，其中一個失效：

- 確認失效節點的狀態：

```
docker ps | grep vmstorage
```

- 測試健康檢查端點：

```
curl http://<failed_vmstorage_host>:8482/health
```





3. 查看日誌：

```
docker logs vmstorage
```

4. 若無法修復，部署新節點並更新配置：

```
docker run -d \
--name vmstorage-new \
-p 8482:8482 \
-v /path/to/storage:/storage \
victoriametrics/vmstorage:<version> \
-retentionPeriod=<days> \
-storageDataPath=/storage
```

5. 測試數據寫入和查詢。

6. 結論

當 **vmstorage** 節點失效時，集群的高可用性機制可減少影響，但仍需快速處理失效節點以恢復集群功能。建議透過以下措施提高穩定性：

- 增加節點數量。
- 啟用數據複製。
- 使用自動發現功能。
- 設置有效的監控與告警規則。

這樣可確保集群在故障發生時仍能穩定運行並快速恢復。



參考"Cluster version.pdf"，以繁體中文詳細說明若2個
vmstorage中的一個失效





當 VictoriaMetrics Cluster 中的兩個 **vmstorage** 節點之一失效時，集群的數據存取功能會部分受影響，但由於集群架構的高可用性設計，集群仍能繼續運行。以下根據《Cluster version.pdf》的內容詳細說明 **vmstorage** 節點部分失效的影響、診斷方法以及處理步驟。

1. 部分 **vmstorage** 節點失效的影響

1. 數據寫入受限：

- **vminsert** 無法將數據寫入失效的 **vmstorage** 節點，可能導致部分數據存取失敗。
- 其他健康的 **vmstorage** 節點需要承擔更多負載。

2. 數據查詢不完整：

- 如果失效節點存有特定數據，**vmselect** 在查詢時可能返回 **partial response**，查詢結果不完整。

3. 性能下降：

- 健康的 **vmstorage** 節點負載增加，可能導致整體性能下降。

4. 數據複製（如果啟用）可能提供冗餘：

- 若啟用了數據複製功能 (`-replicationFactor=N`)，數據可能已分散在多個節點，減少數據丟失風險。

2. 診斷部分 **vmstorage** 節點失效的方法

(1) 驗證健康檢查端點

測試失效的 **vmstorage** 節點的健康狀態：

```
curl http://<failed_vmstorage_host>:8482/health
```

- 返回 `200 OK` 表示節點健康。





- 返回其他狀態或無響應則可能失效。

(2) 檢查 vminsert 的日誌

查看 **vminsert** 的日誌，檢查是否有寫入失敗的錯誤：

```
docker logs vminsert
```

關鍵錯誤：

- **storageNode unreachable。**
- **failed to send data to vmstorage。**

(3) 檢查 vmselect 查詢行為

在 **vmselect** 中執行查詢，檢查是否返回 **partial response**：

```
curl "http://<vmselect_host>:8481/select/0/prometheus/api/v1/query?query=up"
```

- 如果返回部分數據並包含 `{"isPartial": true}`，則表明某些 **vmstorage** 節點無法提供數據。

(4) 查看失效節點的日誌

登入失效節點的伺服器，檢查運行日誌：

```
docker logs vmstorage
```

關注以下錯誤：

- **磁碟空間不足。**
- **記憶體不足 (OOM)。**
- **網路連接失敗。**





3. 部分 vmstorage 節點失效的處理步驟

(1) 確認服務狀態

檢查失效的 **vmstorage** 容器或進程是否正在運行：

```
docker ps | grep vmstorage
```

若未運行，嘗試啟動：

```
docker restart vmstorage
```

若服務未啟動，檢查系統服務：

```
sudo systemctl status vmstorage  
sudo systemctl restart vmstorage
```

(2) 檢查資源狀態

- 確認伺服器的 CPU、記憶體和磁碟空間是否充足：

```
top      # 檢查 CPU 和記憶體使用  
df -h    # 檢查磁碟空間
```

- 若磁碟空間不足，清理無用檔案或擴展存儲。

(3) 測試網路連接

確認失效節點是否能與其他節點正常通信：

```
ping <other_vmstorage_host>  
curl http://<other_vmstorage_host>:8482/metrics
```

(4) 重啟或替換失效節點

如果無法修復失效節點，可考慮替換節點：

- 停止失效的節點。





2. 部署新節點：

```
docker run -d \
--name vmstorage-new \
-p 8482:8482 \
-v /path/to/storage:/storage \
victoriametrics/vmstorage:<version> \
-retentionPeriod=<days> \
-storageDataPath=/storage
```

3. 更新 **vminsert** 和 **vmselect** 的配置以包含新節點：

- **vminsert** :

```
docker restart vminsert --storageNode=<new_vmsorage_host>:8482
```

- **vmselect** :

```
docker restart vmselect --storageNode=<new_vmsorage_host>:8482
```

(5) 測試集群功能

- 測試數據寫入：

```
curl -X POST "http://<vminsert_host>:8480/insert/0/prometheus/api/v1/write"
\
-d '<data_payload>'
```

- 測試數據查詢：

```
curl "http://<vmselect_host>:8481/select/0/prometheus/api/v1/query?
query=up"
```

4. 優化與高可用性配置建議

(1) 啟用數據複製

在 **vminsert** 的啟動參數中啟用數據複製：





```
-replicationFactor=2
```

此配置會將數據複製到多個 **vmstorage** 節點，減少單節點故障對數據的影響。

(2) 增加 **vmstorage** 節點

擴展 **vmstorage** 節點的數量，分散負載，提高集群的穩定性。

(3) 使用自動發現功能

配置自動發現，確保節點列表始終更新：

- 文件方式：

```
-storageNode=file:/path/to/vmstorage-list
```

- DNS SRV 方式：

```
-storageNode=srv+vmstorage-autodiscovery
```

(4) 設置監控與告警

監控 **vmstorage** 的指標，設置告警規則，及時檢測問題：

- vm_rpc_vmstorage_is_reachable**：監控節點是否可達。
- vm_storage_is_read_only**：檢測是否進入只讀模式。

5. 排錯範例：實際操作步驟

假設兩個 **vmstorage** 節點之一失效：

- 確認失效節點的狀態：

```
docker ps | grep vmstorage
```

- 測試失效節點的健康檢查：





```
curl http://<failed_vmsstorage_host>:8482/health
```

3. 查看失效節點的日誌：

```
docker logs vmsstorage
```

4. 若無法修復，部署新節點並更新配置：

```
docker run -d \
--name vmsstorage-new \
-p 8482:8482 \
-v /path/to/storage:/storage \
victoriametrics/vmsstorage:<version> \
-retentionPeriod=<days> \
-storageDataPath=/storage
```

6. 結論

當兩個 **vmsstorage** 節點之一失效時，集群的高可用性機制能確保部分功能正常運行，但需要迅速處理失效節點以恢復完全的數據存取能力。透過以下措施可減少影響：

- 增加節點數量並啟用數據複製。
- 使用自動發現功能管理節點。
- 定期監控節點健康狀況。

這些措施可提高集群穩定性，確保在故障發生時仍能維持正常運作並快速恢復。



參考"Cluster version.pdf"，以繁體中文詳細說明若vmselect
失效





當 **vmselect** 節點失效時，會對 **VictoriaMetrics Cluster** 的數據查詢功能產生影響，但數據的寫入和存儲通常不會受到影響。以下根據《Cluster version.pdf》的內容，詳細說明 **vmselect** 節點失效的影響、診斷方法以及處理步驟。

1. **vmselect** 節點失效的可能影響

1. 查詢功能中斷：

- **vmselect** 是負責從 **vmstorage** 中檢索數據的節點。如果所有 **vmselect** 節點失效，用戶將無法查詢數據。

2. 部分查詢受限：

- 如果集群中有多個 **vmselect** 節點且僅部分失效，負載會重新分配到健康的節點，可能導致性能下降。

3. **Grafana** 顯示異常：

- **Grafana** 可能無法正確顯示監控數據，因為其主要依賴 **vmselect** 進行數據查詢。

2. 診斷 **vmselect** 節點失效的方法

(1) 測試健康檢查端點

訪問失效節點的 `'/health` 接口確認其狀態：

```
curl http://<vmselect_host>:8481/health
```

- 返回 `200 OK` 表示健康。
- 無響應或返回錯誤狀態則表示節點失效。

(2) 檢查 **Grafana** 的行為

- 在 **Grafana** 中測試查詢：





- 如果返回錯誤，如 `Bad Gateway` 或 `timeout`，可能表示 **vmselect** 節點不可用。

(3) 查看 **vmselect** 的日誌

檢查 **vmselect** 節點的運行日誌，診斷失效原因：

```
docker logs vmselect
```

關注以下錯誤：

- 與 **vmstorage** 連接失敗。
- 資源不足（如記憶體不足或 CPU 過載）。
- 請求超時。

(4) 測試與 **vmstorage** 的連線

確認失效的 **vmselect** 是否能與 **vmstorage** 節點正常通信：

```
curl http://<vmstorage_host>:8482/metrics
```

如果無法連接，問題可能出在網路配置。

3. **vmselect** 節點失效的處理步驟

(1) 確認服務狀態

檢查 **vmselect** 容器或服務是否正在運行：

```
docker ps | grep vmselect
```

若未運行，嘗試啟動：

```
docker restart vmselect
```





若服務未啟動，檢查系統服務：

```
sudo systemctl status vmselect  
sudo systemctl restart vmselect
```

(2) 檢查資源使用情況

- 確認伺服器的 CPU 和記憶體使用是否過載：

```
top # 查看 CPU 和記憶體
```

- 如果資源不足，考慮優化配置或升級硬體。

(3) 檢查配置參數

確保 **vmselect** 的啟動參數正確配置，尤其是`-storageNode`是否正確指向 **vmstorage** 節點：

```
docker run -d \  
  --name vmselect \  
  -p 8481:8481 \  
  victoriametrics/vmselect:<version> \  
  -storageNode=<vmstorage_host>:8482
```

(4) 測試查詢功能

執行查詢測試，確認修復效果：

```
curl "http://<vmselect_host>:8481/select/0/prometheus/api/v1/query?query=up"
```

如果查詢返回正確結果，則問題已解決。

(5) 替換失效節點

如果無法修復失效的 **vmselect** 節點，可以部署新節點替代：

- 停止失效的節點。
- 部署新的 **vmselect** 節點：





```
docker run -d \
--name vmselect-new \
-p 8481:8481 \
victoriametrics/vmselect:<version> \
-storageNode=<vmstorage_host>:8482
```

4. 優化與高可用性配置建議

(1) 部署多個 **vmselect** 節點

為實現高可用性，部署多個 **vmselect** 節點，並使用負載均衡器（如 Nginx 或 HAProxy）進行流量分配。

(2) 配置負載均衡器

使用 Nginx 作為負載均衡器，確保查詢流量能自動切換到健康的節點：

```
upstream vmselect {
    server <vmselect1_host>:8481;
    server <vmselect2_host>:8481;
}

server {
    listen 8481;
    location / {
        proxy_pass http://vmselect;
    }
}
```

(3) 監控與告警

設置監控和告警規則，及時檢測 **vmselect** 節點問題：

- **vm_rpc_vmselect_is_reachable**：檢測節點是否可達。
- **vm_http_request_duration_seconds_sum**：監控請求的處理時間。





(4) 啟用自動重啟策略

為 **vmselect** 容器啟用自動重啟策略：

```
docker update --restart always vmselect
```

5. 排錯範例：實際操作步驟

假設有兩個 **vmselect** 節點，其中一個失效：

1. 確認失效節點的狀態：

```
docker ps | grep vmselect
```

2. 測試失效節點的健康檢查：

```
curl http://<failed_vmselect_host>:8481/health
```

3. 查看失效節點的日誌：

```
docker logs vmselect
```

4. 若無法修復，部署新節點替代：

```
docker run -d \
--name vmselect-new \
-p 8481:8481 \
victoriametrics/vmselect:<version> \
-storageNode=<vmstorage_host>:8482
```

6. 結論





當兩個 **vmselect** 節點之一失效時，查詢功能可能部分中斷，但可透過其他健康節點繼續提供服務。為減少影響，建議：

- 部署多個 **vmselect** 節點。
- 使用負載均衡器進行流量分配。
- 定期監控節點健康狀態並設置告警。

這些措施可提高集群的查詢高可用性，確保在單點故障發生時仍能維持穩定的數據查詢能力。



參考"Cluster version.pdf"，以繁體中文詳細說明若2個
vmselect 中的一個失效



當 **VictoriaMetrics Cluster** 中的兩個 **vmselect** 節點之一失效時，集群的數據查詢功能可能部分受到影響，但由於 **vmselect** 的高可用性架構，健康的節點仍可承擔查詢工作。以下根據《Cluster version.pdf》的內容詳細說明 **vmselect** 節點部分失效的影響、診斷方法以及處理步驟。

1. 部分 **vmselect** 節點失效的可能影響

1. 部分查詢重定向：

- 如果使用了負載均衡器，查詢流量會自動重定向到健康的 **vmselect** 節點，但負載可能增加。

2. 性能下降：

- 健康的節點需承擔全部查詢請求，可能導致響應時間增加。

3. 查詢中斷：

- 若數據來源或用戶直接連接到失效的節點，則該查詢可能會中斷或返回錯誤。





2. 診斷部分 **vmselect** 節點失效的方法

(1) 測試健康檢查端點

訪問失效節點的 `/**health**` 接口確認其狀態：

```
curl http://<vmselect_host>:8481/health
```

- 返回 `200 OK` 表示健康。
- 返回其他狀態或無響應則表明該節點失效。

(2) 測試查詢功能

對失效節點執行查詢測試：

```
curl "http://<failed_vmselect_host>:8481/select/0/prometheus/api/v1/query?query=up"
```

- 若返回錯誤或無法連接，說明該節點已失效。

(3) 檢查負載均衡器行為

如果使用了負載均衡器（如 Nginx），檢查其日誌以確認流量是否被正確分配到健康的 **vmselect** 節點。

(4) 查看失效節點的日誌

檢查失效節點的運行日誌以診斷問題：

```
docker logs vmselect
```

關注以下錯誤：

- 與 **vmstorage** 節點連線失敗。
- 資源不足（記憶體不足、CPU 過載等）。





- 請求處理超時。

3. 部分 **vmselect** 節點失效的處理步驟

(1) 確認服務狀態

檢查失效的 **vmselect** 容器或服務是否正在運行：

```
docker ps | grep vmselect
```

若未運行，嘗試啟動：

```
docker restart vmselect
```

若服務未啟動，檢查系統服務：

```
sudo systemctl status vmselect  
sudo systemctl restart vmselect
```

(2) 檢查系統資源

- 確認伺服器的 CPU 和記憶體是否過載：

```
top # 查看 CPU 和記憶體使用情況
```

- 如果資源不足，考慮升級伺服器硬體或優化查詢負載。

(3) 驗證配置

檢查 **vmselect** 的啟動參數是否正確配置：

- **-storageNode** 是否正確指向所有 **vmstorage** 節點。例如：

```
docker run -d \  
--name vmselect \  
...
```





```
-p 8481:8481 \
victoriametrics/vmselect:<version> \
-storageNode=<vmstorage1_host>:8482 \
-storageNode=<vmstorage2_host>:8482
```

(4) 測試修復結果

重啟後測試查詢功能是否恢復：

```
curl "http://<vmselect_host>:8481/select/0/prometheus/api/v1/query?query=up"
```

如果查詢成功，則問題已解決。

(5) 部署新節點（若無法修復）

如果無法修復失效的節點，可考慮部署一個新的 **vmselect** 節點替代：

1. 停止失效的節點。
2. 部署新節點：

```
docker run -d \
--name vmselect-new \
-p 8481:8481 \
victoriametrics/vmselect:<version> \
-storageNode=<vmstorage1_host>:8482 \
-storageNode=<vmstorage2_host>:8482
```

4. 優化與高可用性配置建議

(1) 使用負載均衡器

在所有 **vmselect** 節點前部署負載均衡器（如 Nginx 或 HAProxy），確保流量自動分配到健康節點。範例 Nginx 配置：





```
upstream vmselect {
    server <vmselect1_host>:8481;
    server <vmselect2_host>:8481;
}

server {
    listen 8481;
    location / {
        proxy_pass http://vmselect;
    }
}
```

(2) 增加 **vmselect** 節點

增加 **vmselect** 節點數量以分散查詢流量，提高查詢性能和高可用性。

(3) 啟用自動重啟策略

為所有 **vmselect** 容器設置自動重啟：

```
docker update --restart always vmselect
```

(4) 定期監控與告警

設置監控指標和告警規則，提前檢測潛在問題：

- **vm_rpc_vmselect_is_reachable**：監控節點是否可達。
- **vm_http_request_duration_seconds_sum**：監控查詢處理時間。

5. 排錯範例：實際操作步驟

假設兩個 **vmselect** 節點之一失效：

1. 測試健康檢查：





```
curl http://<failed_vmselect_host>:8481/health
```

2. 查看失效節點日誌：

```
docker logs vmselect
```

3. 嘗試重啟失效節點：

```
docker restart vmselect
```

4. 如果無法修復，部署新節點：

```
docker run -d \
--name vmselect-new \
-p 8481:8481 \
victoriametrics/vmselect:<version> \
-storageNode=<vmstorage1_host>:8482 \
-storageNode=<vmstorage2_host>:8482
```

6. 結論

當兩個 **vmselect** 節點之一失效時，查詢功能會部分受影響，但負載均衡和高可用性設計能減少影響。建議：

1. 部署多個 **vmselect** 節點並使用負載均衡器。
2. 定期監控節點性能和健康狀態。
3. 啟用自動重啟策略以快速恢復失效節點。

透過上述措施，能有效降低單節點故障對集群的影響，確保數據查詢的穩定性和可靠性。



參考"Cluster version.pdf"，以繁體中文詳細說明怎麼debug VictoriaMetrics Cluster





以下是根據《Cluster version.pdf》內容整理的 **VictoriaMetrics Cluster** debug 方法，以繁體中文詳細說明如何排查和解決常見問題，包括 **vminsert**、**vmselect** 和 **vmstorage** 的診斷步驟。

1. 基礎準備

在開始 debug 之前，需確認以下條件：

1. **確保訪問權限**：能夠訪問所有集群節點的運行環境（Docker 或裸機）。
2. **了解系統配置**：熟悉 VictoriaMetrics 的部署架構，包括 `vminsert`、`vmselect` 和 `vmstorage` 的主機和端口配置。
3. **工具準備**：
 - 使用 `curl` 測試健康檢查和指標接口。
 - 日誌檢視工具（如 `docker logs` 或 `journalctl`）。

2. 常見問題與排查步驟

(1) 數據寫入問題

問題描述：

- **vmagent** 或數據來源無法將數據寫入集群。

排查步驟：

1. 檢查 **vminsert** 健康狀態 測試 `vminsert` 的健康檢查端點：

```
curl http://<vminsert_host>:8480/health
```





- 返回 `200 OK` 表示正常。
- 如果無響應或返回錯誤，需檢查 `vminsert` 是否正在運行：

```
docker ps | grep vminsert
```

- 若未運行，嘗試重啟：

```
docker restart vminsert
```

2. 檢查數據來源配置 確認數據來源的 `remoteWrite.url` 是否正確指向 **vminsert**：

```
http://<vminsert_host>:8480/insert/0/prometheus/api/v1/write
```

3. 檢查 **vminsert** 日誌 查看 `vminsert` 是否報告與 `vmstorage` 的連線問題：

```
docker logs vminsert
```

4. 測試數據寫入 使用 `curl` 模擬數據寫入：

```
curl -X POST "http://<vminsert_host>:8480/insert/0/prometheus/api/v1/write"
\
-d '<data_payload>'
```

- 如果返回錯誤，檢查 `vmstorage` 是否健康（參見下一步）。

(2) 數據存儲問題

問題描述：

- **vminsert** 無法將數據存儲到 **vmstorage**，或 **vmstorage** 無法正常運行。

排查步驟：

1. 檢查 **vmstorage** 健康狀態 測試 `vmstorage` 的健康檢查端點：



```
curl http://<vmstorage_host>:8482/health
```

- 返回 `200 OK` 表示健康。
- 無響應則需檢查 `vmstorage` 是否運行：

```
docker ps | grep vmstorage
```

2. 檢查資源使用情況

- 確認磁碟空間是否不足：

```
df -h
```

- 檢查 CPU 和記憶體使用：

```
top
```

3. 檢查 vmstorage 日誌 查看 `vmstorage` 的運行日誌：

```
docker logs vmstorage
```

- 若出現磁碟空間錯誤，需清理或擴展存儲。
- 若報告連線失敗，檢查網路配置。

4. 測試與 vminspect 的連線 確認 `vminspect` 能連接到所有 `vmstorage` 節點：

```
curl http://<vmstorage_host>:8482/metrics
```

(3) 數據查詢問題

問題描述：

- 查詢數據返回錯誤或結果不完整，可能涉及 **vmselect** 節點。





排查步驟：

1. 檢查 **vmselect** 健康狀態 測試 `vmselect` 的健康檢查端點：

```
curl http://<vmselect_host>:8481/health
```

- 返回 `200 OK` 表示健康。
- 無響應則需檢查 `vmselect` 是否運行：

```
docker ps | grep vmselect
```

2. 檢查 **vmselect** 與 **vmstorage** 的連線 確認 `vmselect` 能正確訪問所有 `vmstorage` 節點：

```
curl http://<vmstorage_host>:8482/metrics
```

3. 檢查 **vmselect** 日誌 查看 `vmselect` 的運行日誌，查找查詢超時或連線錯誤信息：

```
docker logs vmselect
```

4. 測試查詢功能 使用 `curl` 測試查詢功能：

```
curl "http://<vmselect_host>:8481/select/0/prometheus/api/v1/query?query=up"
```

- 若查詢失敗，檢查 PromQL 語法和數據範圍。

(4) 整體性能問題

問題描述：

- 集群性能下降，如查詢延遲增加、寫入速度減慢等。





排查步驟：

1. 檢查指標數據 在 `vmselect` 和 `vminsert` 上檢查性能相關指標：

- **vm_http_request_duration_seconds_sum**：請求處理時間。
- **vm_http_requests_total**：請求總數。
- **vm_rpc_vmstorage_is_reachable**：檢測節點連線狀態。

2. 監控資源使用

- 檢查所有節點的資源使用情況，包括 CPU、記憶體和磁碟。
3. 優化 PromQL 查詢 確認查詢是否過於複雜或消耗資源，優化 PromQL 語法。
-

3. 高效 Debug 技巧

1. 分層排查：

- 從數據來源 (**vmagent**) 到數據存儲 (**vmstorage**) 再到數據查詢 (**vmselect**)，逐步檢查。

2. 使用負載均衡：

- 為多個 `vmselect` 和 `vminsert` 節點配置負載均衡器（如 Nginx），確保查詢和寫入的高可用性。

3. 啟用指標和告警：

- 使用 Prometheus 和 Grafana 監控集群的健康狀況，及時發現問題。
-

4. 結論





Debug **VictoriaMetrics Cluster** 時，需先確定問題發生的位置（寫入、存儲或查詢），再根據節點類型逐步排查。通過健康檢查接口、日誌檢視及性能指標監控，可以快速找到並解決問題。同時，建議啟用高可用性配置（多節點部署和負載均衡）以減少故障影響。



參考"Cluster version.pdf"，以繁體中文詳細說明若改用 docker-compose建立cluster VictoriaMetrics



以下是參考《Cluster version.pdf》內容，詳細說明如何使用 **Docker Compose** 建立 **VictoriaMetrics Cluster** 的步驟。

1. 準備環境

(1) 安裝必要軟體

確保已安裝以下工具：

- **Docker**：提供容器化運行環境。
- **Docker Compose**：簡化多容器部署的工具。

(2) 建立專案目錄

創建一個專案目錄以存放 `docker-compose.yml` 配置文件及相關資料：

```
mkdir victoria-metrics-cluster  
cd victoria-metrics-cluster
```

2. 建立 docker-compose.yml 文件





在專案目錄內新建 `docker-compose.yml` 文件，定義 **vminsert**、**vmselect** 和 **vmstorage** 的服務配置。以下是一個基本的範例：

```
version: '3.7'

services:
  vmstorage:
    image: victoriametrics/vmstorage:v1.93.2
    container_name: vmstorage
    ports:
      - "8482:8482"
    volumes:
      - ./storage:/storage
    command:
      - "-retentionPeriod=30"
      - "-storageDataPath=/storage"
    restart: always

  vminsert:
    image: victoriametrics/vminsert:v1.93.2
    container_name: vminsert
    ports:
      - "8480:8480"
    depends_on:
      - vmstorage
    command:
      - "-storageNode=vmstorage:8482"
    restart: always

  vmselect:
    image: victoriametrics/vmselect:v1.93.2
    container_name: vmselect
    ports:
      - "8481:8481"
    depends_on:
      - vmstorage
    command:
      - "-storageNode=vmstorage:8482"
    restart: always
```





文件解釋：

1. `vmstorage`：

- 負責數據的存儲。
- 使用 `-retentionPeriod` 設置數據保留天數（此處為 30 天）。
- 將本地目錄 `./storage` 掛載到容器內的 `/storage`。

2. `vminsert`：

- 負責數據寫入，並將數據傳遞至 `vmstorage`。
- 使用 `-storageNode` 指向 `vmstorage` 節點。

3. `vmselect`：

- 負責數據查詢，並從 `vmstorage` 中檢索數據。
- 使用 `-storageNode` 指向 `vmstorage` 節點。

3. 啟動集群

在專案目錄中運行以下命令啟動集群：

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

驗證服務運行：

1. 檢查容器狀態：

```
docker-compose ps
```

確認所有服務的狀態為 `Up`。

2. 訪問健康檢查端點：



- **vminsert** : `http://localhost:8480/health`
- **vmselect** : `http://localhost:8481/health`
- **vmstorage** : `http://localhost:8482/health` 若返回 `200 OK` 表示服務正常。

4. 測試集群功能

(1) 測試數據寫入

使用 `curl` 模擬數據寫入至 **vminsert** :

```
curl -X POST "http://localhost:8480/insert/0/prometheus/api/v1/write" \
-d '<data_payload>'
```

(2) 測試數據查詢

通過 **vmselect** 查詢數據 :

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

5. 擴展配置 (多節點)

如果需要支持更高的負載，可以擴展 **vminsert** 和 **vmselect** 節點。

更新 **docker-compose.yml** 文件：

新增多個 `vminsert` 和 `vmselect` 節點，並使用負載均衡器分配流量。

範例：

```
vminsert1:
  image: victoriametrics/vminsert:v1.93.2
  container_name: vminsert1
```





```
ports:
  - "8483:8480"
depends_on:
  - vmstorage
command:
  - "-storageNode=vmstorage:8482"
restart: always

vmselect1:
  image: victoriametrics/vmselect:v1.93.2
  container_name: vmselect1
  ports:
    - "8484:8481"
depends_on:
  - vmstorage
command:
  - "-storageNode=vmstorage:8482"
restart: always
```

6. 停止與清理

停止集群：

```
docker-compose down
```

清理數據（可選）：

刪除存儲目錄內的數據以重置環境：

```
rm -rf ./storage
```

7. 建議與最佳實踐



1. **資料持久化**：確保 `./storage` 目錄存放於可靠的存儲設備上，以防數據丟失。

2. **監控和告警**：

- 使用 **Prometheus** 監控各服務的健康狀況和性能。
- 在 **Grafana** 中設置儀表板，視覺化數據指標。

3. **高可用性**：

- 部署多個節點的 **vminsert** 和 **vmselect**，並使用負載均衡器（如 Nginx）分配流量。

4. **定期備份**：定期備份存儲目錄以防止數據損失。

以上步驟可協助使用 **Docker Compose** 快速部署和管理 **VictoriaMetrics Cluster**，滿足監控與數據存儲需求。



在 **Kubernetes** 上使用 **StatefulSet** 部署 **vminsert** (**VictoriaMetrics Cluster**)

1. 概述

在 **Kubernetes (K8s)** 中，`vminsert` 負責接收監控數據並將其寫入 `vmstorage`。為確保 `vminsert` 在 K8s 中穩定運行並具備持久性，我們將使用 **StatefulSet** 進行部署，以確保：

- `vminsert` 節點具備 **固定的 DNS 名稱**，便於 `vmagent` 發送數據。





- 多個 `vminsert` 副本 可水平擴展，提高吞吐量。
 - 持久化 `vminsert` 的日誌和配置（儘管 `vminsert` 本身為無狀態應用）。
-

2. 建立 StatefulSet 配置

我們將使用 `StatefulSet` 部署 **vminsert**，並確保它能與 **vmstorage** 正確連接。

(1) 建立 ConfigMap

ConfigMap 用於存儲 `vminsert` 的啟動參數，如 `storageNode` 的 URL。

創建 `vminsert-configmap.yaml`：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: vminsert-config
  namespace: victoria-metrics
data:
  vminsert-flags: |
    -storageNode=vmstorage-0.vmstorage:8482
    -storageNode=vmstorage-1.vmstorage:8482
```

部署 ConfigMap：

```
kubectl apply -f vminsert-configmap.yaml
```

(2) 建立 StatefulSet

創建 `vminsert-statefulset.yaml`：





```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: vminsert
  namespace: victoria-metrics
spec:
  serviceName: "vminsert"
  replicas: 2
  selector:
    matchLabels:
      app: vminsert
  template:
    metadata:
      labels:
        app: vminsert
    spec:
      containers:
        - name: vminsert
          image: victoriametrics/vminsert:v1.93.2
          args:
            - "--storageNode=$(STORAGE_NODES)"
      env:
        - name: STORAGE_NODES
          valueFrom:
            configMapKeyRef:
              name: vminsert-config
              key: vminsert-flags
      ports:
        - containerPort: 8480
          name: http
      readinessProbe:
        httpGet:
          path: /health
          port: 8480
        initialDelaySeconds: 5
        periodSeconds: 10
      livenessProbe:
        httpGet:
          path: /health
          port: 8480
```





```
    initialDelaySeconds: 15
    periodSeconds: 20
  resources:
    requests:
      cpu: "500m"
      memory: "512Mi"
    limits:
      cpu: "1000m"
      memory: "1Gi"
  terminationGracePeriodSeconds: 10
```

此 `StatefulSet` 配置：

- `replicas: 2`：創建兩個 `vminsert` 副本。
- `serviceName: "vminsert"`：確保 `vminsert` 節點有固定的 DNS 名稱。
- 環境變數 `STORAGE_NODES`：從 `ConfigMap` 讀取 `vmstorage` 節點地址。
- 健康檢查 (`livenessProbe` & `readinessProbe`)：確保 `vminsert` 運行正常。

部署 StatefulSet：

```
kubectl apply -f vminsert-statefulset.yaml
```

(3) 建立 Headless Service

為 `vminsert` 創建一個無頭服務（Headless Service），以便 `vmagent` 能夠動態發現 `vminsert` 節點。

創建 `vminsert-service.yaml`：

```
apiVersion: v1
kind: Service
metadata:
  name: vminsert
  namespace: victoria-metrics
```





```
spec:  
  ports:  
    - port: 8480  
      targetPort: 8480  
      protocol: TCP  
      name: http  
  clusterIP: None  
  selector:  
    app: vminsert
```

部署 Service：

```
kubectl apply -f vminsert-service.yaml
```

3. 測試與驗證

(1) 驗證 Pod 狀態

檢查 `vminsert` 是否成功部署：

```
kubectl get pods -n victoria-metrics
```

應該看到類似輸出的結果：

NAME	READY	STATUS	RESTARTS	AGE
vminsert-0	1/1	Running	0	10s
vminsert-1	1/1	Running	0	10s

(2) 測試 vminsert 健康狀態

```
kubectl port-forward svc/vminsert 8480 -n victoria-metrics  
curl http://localhost:8480/health
```





返回 `200 OK` 則表示 `vminsert` 正常運行。

(3) 測試數據寫入

模擬 **vmagent** 或 Prometheus 進行數據寫入：

```
curl -X POST "http://vminsert-0.vminsert:8480/insert/0/prometheus/api/v1/write"
\ 
-d '<data_payload>'
```

4. 擴展與最佳實踐

(1) 水平擴展 vminsert

如果需要處理更大流量，可以調整 `replicas` 參數：

```
kubectl scale statefulset vminsert --replicas=3 -n victoria-metrics
```

(2) 使用 `vmagent` 自動發現 vminsert

在 `vmagent` 配置文件中，使用 `kubernetes_sd_configs` 自動發現 `vminsert`：

```
scrape_configs:
- job_name: 'vminsert'
  kubernetes_sd_configs:
    - role: endpoints
      namespaces:
        names:
        - victoria-metrics
```

(3) 監控與告警

可以使用 **Prometheus + Grafana** 監控 `vminsert`：

- 監控指標：



- `vm_http_requests_total` (HTTP 請求總數)
- `vm_rpc_vmsstorage_is_reachable` (vmsstorage 連線狀態)

5. 停止與刪除

如果需要刪除 `vminsert`：

```
kubectl delete statefulset vminsert -n victoria-metrics  
kubectl delete service vminsert -n victoria-metrics  
kubectl delete configmap vminsert-config -n victoria-metrics
```

6. 總結

組件	角色
StatefulSet	確保 `vminsert` 具備固定 DNS 名稱和擴展能力。
ConfigMap	設定 `storageNode`，確保 `vminsert` 連接 `vmsstorage`。
Headless Service	確保 `vmagent` 能自動發現 `vminsert` 節點。
水平擴展	可根據需求增加 `vminsert` 節點，提高寫入能力。

透過這些步驟，即可在 **Kubernetes** 上透過 **StatefulSet** 部署 **vminsert**，並確保其高可用性與可擴展性！



詳細說明:怎麼改成k8s上，透過statefulset部屬vmagent





在 Kubernetes 上使用 StatefulSet 部署 vmagent (VictoriaMetrics Cluster)

1. 概述

在 **Kubernetes (K8s)** 環境中，`vmagent` 負責從監控目標（如 **Prometheus**、**Node Exporter**、**cAdvisor**）收集數據，並將其發送至 `vminsert`，以寫入 **VictoriaMetrics Cluster**。

使用 **StatefulSet** 部署 `vmagent` 具有以下優勢：

- **具備固定的 DNS 名稱**，便於 `vmagent` 在多副本運行時能被可靠定位。
- **多副本支持（可水平擴展）**，增加 `vmagent` 的抓取能力，避免單點故障。
- **持久化存儲** `vmagent` 積存數據，以防止數據丟失。

2. 建立 Kubernetes 配置文件

我們將使用 **ConfigMap** 存儲 `vmagent` 的 **scrape** 配置，並使用 **StatefulSet** 部署 `vmagent`。

(1) 創建 ConfigMap

ConfigMap 用於存儲 `vmagent` 的抓取目標（如 `Node Exporter`、`cAdvisor`、`kubelet`）。

創建 `vmagent-configmap.yaml`：





```
apiVersion: v1
kind: ConfigMap
metadata:
  name: vmagent-config
  namespace: victoria-metrics
data:
  vmagent.yaml: |
    global:
      scrape_interval: 15s
      scrape_timeout: 10s
      scrape_configs:
        - job_name: 'kubernetes-nodes'
          kubernetes_sd_configs:
            - role: node
        - job_name: 'kubernetes-pods'
          kubernetes_sd_configs:
            - role: pod
        - job_name: 'vminsert'
          static_configs:
            - targets: ['vminsert.victoria-metrics.svc.cluster.local:8480']
```

部署 ConfigMap：

```
kubectl apply -f vmagent-configmap.yaml
```

(2) 創建 StatefulSet

創建 `vmagent-statefulset.yaml`：

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: vmagent
  namespace: victoria-metrics
```





```
spec:
  serviceName: "vmagent"
  replicas: 2
  selector:
    matchLabels:
      app: vmagent
  template:
    metadata:
      labels:
        app: vmagent
    spec:
      volumes:
        - name: config-volume
          configMap:
            name: vmagent-config
      containers:
        - name: vmagent
          image: victoriametrics/vmagent:v1.93.2
          args:
            - "-remoteWrite.url=http://vminsert.victoria-metrics.svc.cluster.local:8480/insert/0/prometheus/api/v1/write"
            - "-promscrape.config=/etc/vmagent/vmagent.yml"
          volumeMounts:
            - name: config-volume
              mountPath: /etc/vmagent/
      ports:
        - containerPort: 8429
          name: http
      readinessProbe:
        httpGet:
          path: /targets
          port: 8429
        initialDelaySeconds: 5
        periodSeconds: 10
      livenessProbe:
        httpGet:
          path: /metrics
          port: 8429
        initialDelaySeconds: 15
        periodSeconds: 20
```





```
resources:
  requests:
    cpu: "500m"
    memory: "512Mi"
  limits:
    cpu: "1000m"
    memory: "1Gi"
terminationGracePeriodSeconds: 10
```

說明：

- `replicas: 2`：啟動兩個 `vmagent` 副本。
- `--remoteWrite.url`：指定 `vminsert` 節點作為數據存儲目標。
- `--promscrape.config=/etc/vmagent/vmagent.yaml`：從 `ConfigMap` 加載 `vmagent` 配置。
- **健康檢查 (`readinessProbe` & `livenessProbe`)**：
 - `/targets` 用於檢查 `vmagent` 是否正確抓取數據。
 - `/metrics` 用於監控 `vmagent` 的健康狀態。

部署 StatefulSet：

```
kubectl apply -f vmagent-statefulset.yaml
```

(3) 創建 Headless Service

為 `vmagent` 創建 **無頭服務 (Headless Service)**，確保 `vmagent` 節點能夠被 `kubernetes_sd_configs` 自動發現。

創建 `vmagent-service.yaml`：

```
apiVersion: v1
kind: Service
```





```
metadata:  
  name: vmagent  
  namespace: victoria-metrics  
spec:  
  ports:  
    - port: 8429  
      targetPort: 8429  
      protocol: TCP  
      name: http  
  clusterIP: None  
  selector:  
    app: vmagent
```

部署 Service：

```
kubectl apply -f vmagent-service.yaml
```

3. 測試與驗證

(1) 確認 **vmagent** 是否成功運行

```
kubectl get pods -n victoria-metrics
```

應該看到類似的輸出：

NAME	READY	STATUS	RESTARTS	AGE
vmagent-0	1/1	Running	0	10s
vmagent-1	1/1	Running	0	10s

(2) 測試 **vmagent** 健康狀態





```
kubectl port-forward svc/vmagent 8429 -n victoria-metrics  
curl http://localhost:8429/metrics
```

如果返回 Prometheus 指標，表示 `vmagent` 正常運行。

(3) 測試數據寫入

模擬數據寫入到 `vminsert`：

```
curl -X POST "http://vminsert.victoria-  
metrics.svc.cluster.local:8480/insert/0/prometheus/api/v1/write" \  
-d '<data_payload>'
```

4. 擴展與最佳實踐

(1) 水平擴展 vmagent

如需增加 `vmagent` 副本數量，提高數據抓取能力：

```
kubectl scale statefulset vmagent --replicas=3 -n victoria-metrics
```

(2) 監控 vmagent

可在 **Prometheus** 中監控以下 `vmagent` 指標：

- `vm_promscrape_targets`：監控抓取的目標數量。
- `vm_promscrape_scrapes_total`：監控成功的數據抓取次數。

(3) 使用 `kubernetes_sd_configs` 自動發現 vmagent

在 Prometheus 配置中，啟用 `kubernetes_sd_configs` 以自動發現 `vmagent`：





```
scrape_configs:
  - job_name: 'vmagent'
    kubernetes_sd_configs:
      - role: endpoints
        namespaces:
          names:
            - victoria-metrics
```

(4) 設置告警

可以設置 `vmagent` 失效的告警：

```
ALERT vmagent_down
  IF up{job="vmagent"} == 0
  FOR 5m
  LABELS {severity="critical"}
  ANNOTATIONS {
    summary = "vmagent is down",
    description = "No vmagent instance has been up for the last 5 minutes."
}
```

5. 停止與刪除

如果需要刪除 `vmagent`：

```
kubectl delete statefulset vmagent -n victoria-metrics
kubectl delete service vmagent -n victoria-metrics
kubectl delete configmap vmagent-config -n victoria-metrics
```





6. 總結

組件	角色
StatefulSet	確保 `vmagent` 具備固定 DNS 名稱與擴展能力。
ConfigMap	存儲 `vmagent` 抓取目標的設定。
Headless Service	確保 Prometheus 能自動發現 `vmagent`。
水平擴展	增加 `vmagent` 副本，提高數據抓取能力。

透過這些步驟，即可在 **Kubernetes** 上透過 **StatefulSet** 部署 **vmagent**，並確保其高可用性與可擴展性！ 



詳細說明: k8s上部屬vminsert，應該使用deployment還是statefulse



Kubernetes 上部署 `vminsert`，應該使用 `Deployment` 還是 `StatefulSet`？

在 Kubernetes (K8s) 部署 **VictoriaMetrics Cluster** 時，`vminsert` 負責接收監控數據並將其分發到 **vmstorage** 節點。根據 `vminsert` 的特性和需求，應該選擇 **Deployment** 還是 **StatefulSet**？以下是詳細分析。

1. `vminsert` 需要 `StatefulSet` 嗎？

`StatefulSet` 通常適用於：





1. 需要固定 DNS 名稱的應用 (如 `vmstorage`)。
2. 需要有序啟動、關閉的應用 (如 `vmstorage`)。
3. 需要持久化存儲的應用 (如 `vmstorage`)。

但 `vminsert` 不需要這些特性，因此應該使用 `Deployment`，而非 `StatefulSet`。

2. `Deployment` 適合 `vminsert` 的原因

比較項目	`Deployment`	`StatefulSet`	`vminsert` 適用性
擴展性	✓ 支持動態擴展	⚠ 可能需要手動調整 `storageNode`	✓ `vminsert` 需要
固定 DNS	✗ Pod 名稱隨機變更	✓ 每個 Pod 有固定 DNS	✗ `vminsert` 不需要
有序啟動	✗ 無序啟動	✓ 確保有序啟動	✗ `vminsert` 不需要
持久存儲	✗ 無狀態	✓ 需要存儲	✗ `vminsert` 不需要
負載均衡	✓ Kubernetes 內建	✓ 需額外配置 Service	✓ `vminsert` 需要

✓ 結論

- `vminsert` 無需持久存儲，無需有序啟動，無需固定 DNS，因此應該使用 `Deployment`。
 - `Deployment` 可以讓 `vminsert` 動態擴展，快速滾動更新，適合高併發寫入場景。
-

3. 使用 `Deployment` 部署 `vminsert`

在 K8s 上，使用 `Deployment` 來部署 `vminsert`，確保 高可用性 (HA) 和 負載均衡。





(1) 建立 ConfigMap

使用 **ConfigMap** 存儲 `vminsert` 的參數設定，如 `vmstorage` 節點列表。

📌 建立 `vminsert-configmap.yaml`

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: vminsert-config
  namespace: victoria-metrics
data:
  vminsert-flags: |
    -storageNode=vmstorage-0.vmstorage:8482
    -storageNode=vmstorage-1.vmstorage:8482
```

◆ 部署 ConfigMap

```
kubectl apply -f vminsert-configmap.yaml
```

(2) 建立 `Deployment`

📌 建立 `vminsert-deployment.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vminsert
  namespace: victoria-metrics
spec:
  replicas: 3
  selector:
    matchLabels:
      app: vminsert
```





```
template:
  metadata:
    labels:
      app: vminsert
  spec:
    containers:
      - name: vminsert
        image: victoriametrics/vminsert:v1.93.2
        args:
          - "--storageNode=$(STORAGE_NODES)"
        env:
          - name: STORAGE_NODES
            valueFrom:
              configMapKeyRef:
                name: vminsert-config
                key: vminsert-flags
        ports:
          - containerPort: 8480
            name: http
        readinessProbe:
          httpGet:
            path: /health
            port: 8480
            initialDelaySeconds: 5
            periodSeconds: 10
        livenessProbe:
          httpGet:
            path: /health
            port: 8480
            initialDelaySeconds: 15
            periodSeconds: 20
        resources:
          requests:
            cpu: "500m"
            memory: "512Mi"
          limits:
            cpu: "1000m"
            memory: "1Gi"
        terminationGracePeriodSeconds: 10
```





◆ 部署 `vminsert`

```
kubectl apply -f vminsert-deployment.yaml
```

(3) 建立 Service

由於 `vminsert` 是無狀態應用，應該使用 **ClusterIP Service** 來讓 `vmagent` 連接。

📌 建立 `vminsert-service.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: vminsert
  namespace: victoria-metrics
spec:
  ports:
    - port: 8480
      targetPort: 8480
      protocol: TCP
      name: http
  selector:
    app: vminsert
```

◆ 部署 Service

```
kubectl apply -f vminsert-service.yaml
```

4. 測試與驗證





(1) 確認 Pod 運行

```
kubectl get pods -n victoria-metrics
```

應該看到：

NAME	READY	STATUS	RESTARTS	AGE
vminsert-7f94b5c6d4-6q9w2	1/1	Running	0	10s
vminsert-7f94b5c6d4-hm2v8	1/1	Running	0	10s
vminsert-7f94b5c6d4-1x8w7	1/1	Running	0	10s

(2) 測試健康狀態

```
kubectl port-forward svc/vminsert 8480 -n victoria-metrics
curl http://localhost:8480/health
```

如果返回 `200 OK`，表示 `vminsert` 正常運行。

(3) 測試數據寫入

模擬 `vmagent` 或 Prometheus 發送數據：

```
curl -X POST "http://vminsert.victoria-
metrics.svc.cluster.local:8480/insert/0/prometheus/api/v1/write" \
-d '<data_payload>'
```

5. 擴展與最佳實踐

(1) 水平擴展

如果需要增加 `vminsert` 副本：



```
kubectl scale deployment vminsert --replicas=5 -n victoria-metrics
```

(2) 監控 `vminsert`

在 **Prometheus** 監控以下指標：

- `vm_http_requests_total`：監控寫入請求總數。
- `vm_http_request_duration_seconds_sum`：監控請求處理時間。

(3) 設置負載均衡

若 `vmagent` 需要負載均衡 `vminsert`：

```
remoteWrite:  
  - url: http://vminsert.victoria-  
    metrics.svc.cluster.local:8480/insert/0/prometheus/api/v1/write
```

6. 總結

部署方式	適用於 vminsert 嗎？	原因
Deployment	適用	無狀態，可彈性擴展，滾動更新
StatefulSet	不適用	`vminsert` 無需持久存儲或固定 DNS

📌 `vminsert` 應使用 `Deployment`，確保高可用性、擴展性與負載均衡。🚀



詳細說明:vminsert 需要 StatefulSet 嗎





vminsert 是否需要 StatefulSet？

1. `vminsert` 角色簡介

在 **VictoriaMetrics Cluster** 架構中：

- `vminsert` 負責接收來自 `vmagent`、Prometheus `remoteWrite` 等數據來源的寫入請求，並將數據分發到 **vmstorage** 進行存儲。
- `vminsert` 本身不存儲數據，它只是 **數據轉發節點**，並且可以部署多個實例來實現負載均衡和高可用性 (HA)。

2. `StatefulSet` 適用於哪些情境？

Kubernetes 的 `StatefulSet` 主要適用於 **需要穩定標識、持久存儲及有序啟動/關閉的應用**：

適用場景	StatefulSet 是否適用	vminsert 是否符合？
需要固定 DNS 名稱	✓ 是	✗ `vminsert` 不需要
需要有序啟動/關閉	✓ 是	✗ `vminsert` 無需有序啟動
需要持久存儲	✓ 是	✗ `vminsert` 無狀態
需要集群成員唯一性	✓ 是	✗ `vminsert` 任何副本可互換
可動態擴展	✗ 需要手動更新 `storageNode` 配置	✓ `vminsert` 應自動擴展





3. `vminsert` 需要 StatefulSet 嗎？

🚫 不需要 `StatefulSet`，應該使用 **Deployment**。

原因如下：

1. 無需固定 DNS 名稱

- `vminsert` 只需負責數據寫入，它與 `vmagent`、或其他寫入來源無需固定 DNS 解析。
- `vmagent` 只要能透過 **負載均衡 (Service)** 找到 `vminsert` 節點，即可正常運作。

2. 無需有序啟動或關閉

- `vminsert` 可以即時啟動或關閉，因為 **它不存儲任何狀態**。
- `StatefulSet` 會確保 Pod 按序啟動/關閉，但這對 `vminsert` 來說 **沒有必要**。

3. 無需持久化存儲

- `vminsert` 不會存儲數據，它只是負責將數據轉發給 `vmstorage`，因此 **不需要 Volume Claim**。

4. 需要動態擴展

- `Deployment` 可自由 **擴展 replicas 副本數量**，Pod 之間完全無狀態，任意副本都可以接受數據。
- `StatefulSet` 擴展時，每個 Pod 名稱固定（如 `vminsert-0`、`vminsert-1`），並且可能需要手動調整 `storageNode`，不適合 `vminsert`。

5. 負載均衡

- `vminsert` 需要透過 **Service** 進行負載均衡，而不是單獨連接到某個特定 Pod。
- `Deployment` 搭配 **ClusterIP Service**，可讓 `vmagent`、或其他客戶端通過 `vminsert` 的 **Service** 均衡流量。





4. `Deployment` 如何適用於 `vminsert` ?

因為 `vminsert` 是 **無狀態 (stateless)**，適合 `Deployment` 進行管理。

📌 `Deployment` 主要特點：

- **動態擴展**：可以隨時 **scale up/down** `vminsert` 副本。
- **無固定 Pod 名稱**：Pod 會根據副本數動態分配，如 `vminsert-xyz12`。
- **滾動更新**：當更新 `vminsert` 鏡像時，可以**逐步更新**而不影響服務。
- **負載均衡**：`vminsert` 可以透過 Kubernetes `Service` 對外提供單一訪問點。

5. `Deployment` 部署 `vminsert`

📌 建立 `vminsert-deployment.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vminsert
  namespace: victoria-metrics
spec:
  replicas: 3  # 允許水平擴展
  selector:
    matchLabels:
      app: vminsert
  template:
    metadata:
      labels:
        app: vminsert
    spec:
      containers:
        - name: vminsert
          image: victoriametrics/vminsert:v1.93.2
          args:
            - "--storageNode=vmstorage-0.vmstorage:8482"
```





```
- "--storageNode=vmstorage-1.vmstorage:8482"
ports:
- containerPort: 8480
  name: http
readinessProbe:
  httpGet:
    path: /health
    port: 8480
  initialDelaySeconds: 5
  periodSeconds: 10
livenessProbe:
  httpGet:
    path: /health
    port: 8480
  initialDelaySeconds: 15
  periodSeconds: 20
resources:
  requests:
    cpu: "500m"
    memory: "512Mi"
  limits:
    cpu: "1000m"
    memory: "1Gi"
```

◆ 部署 `vminsert`

```
kubectl apply -f vminsert-deployment.yaml
```

🚀 建立 `Service`

```
apiVersion: v1
kind: Service
metadata:
  name: vminsert
  namespace: victoria-metrics
spec:
  ports:
    - port: 8480
```





```
targetPort: 8480
protocol: TCP
name: http
selector:
app: vminsert
```

◆ 部署 Service

```
kubectl apply -f vminsert-service.yaml
```

6. 總結

🚫 `vminsert` 不需要 `StatefulSet`，應該使用 `Deployment`，因為：

條件	`StatefulSet`	`Deployment`	`vminsert` 需求
無狀態應用	✗ 否	✓ 是	✓ `vminsert` 是無狀態
固定 Pod 名稱	✓ 需要	✗ 不需要	✗ `vminsert` 不需要
可水平擴展	⚠ 手動更新 `storageNode`	✓ 可動態擴展	✓ `vminsert` 需要
負載均衡	⚠ 需手動配置	✓ Kubernetes 內建	✓ `vminsert` 需要
滾動更新	✗ 難以實現	✓ 支持滾動更新	✓ `vminsert` 需要

🚫 最終結論：`vminsert` 適合 **Deployment**，不應該使用 **StatefulSet**。

✓ 使用 Deployment

- 動態擴展：`kubectl scale deployment vminsert --replicas=5`
- 負載均衡：`vmagent` 透過 `Service` 訪問 `vminsert`
- 滾動更新：無需停機即可更新 `vminsert` 版本





🚀 推薦：

- `vminsert` 使用 **Deployment**，搭配 **Service** 進行負載均衡，確保高可用性與擴展性。
- `vmstorage` 使用 **StatefulSet**，因為它需要 **固定 DNS** 和 **持久存儲**。

這樣的架構能夠確保 **VictoriaMetrics Cluster** 具有最佳的擴展性和穩定性！ 🚀



公司的pipeline都是使用statefuleset來部署，請問我要怎麼改成用deployment



如何將 `vminsert` 從 **StatefulSet** 轉換為 **Deployment**

1. 背景與需求

目前公司的 CI/CD pipeline 使用 `StatefulSet` 來部署 `vminsert`，但我們已確認 `vminsert` 不需要 **StatefulSet**，而是應該使用 `Deployment` 來獲得：

- 更好的 **動態擴展性** (Scaling)。
- 更快的 **滾動更新** (Rolling Update)。
- 無需 **固定 DNS 名稱** (StatefulSet 會固定 Pod 名稱，而 `vminsert` 不需要)。
- 更好的 **負載均衡** (可以透過 `Service` 進行流量分配)。

2. 轉換方法

我們的目標是 **將 `StatefulSet` 替換為 `Deployment`**，並確保：

1. **不影響現有的 CI/CD pipeline。**
2. **不影響 `vminsert` 和 `vmstorage` 的數據寫入流程。**
3. **確保負載均衡正常運作。**





3. 原 `StatefulSet` 配置

目前 `vminsert` 的 `StatefulSet` 可能類似這樣：

📌 原始 `StatefulSet` 配置

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: vminsert
  namespace: victoria-metrics
spec:
  serviceName: "vminsert"
  replicas: 2
  selector:
    matchLabels:
      app: vminsert
  template:
    metadata:
      labels:
        app: vminsert
    spec:
      containers:
        - name: vminsert
          image: victoriametrics/vminsert:v1.93.2
          args:
            - "--storageNode=vmstorage-0.vmstorage:8482"
            - "--storageNode=vmstorage-1.vmstorage:8482"
          ports:
            - containerPort: 8480
              name: http
```

📌 原 `Service` (Headless Service)

```
apiVersion: v1
kind: Service
```





```
metadata:
  name: vminsert
  namespace: victoria-metrics
spec:
  clusterIP: None # 無頭服務 (Headless Service)
  ports:
    - port: 8480
      targetPort: 8480
      protocol: TCP
      name: http
  selector:
    app: vminsert
```

4. 轉換為 Deployment

(1) 刪除 StatefulSet

在轉換前，確保 StatefulSet 中的 `vminsert` 沒有影響到數據流，然後刪除它：

```
kubectl delete statefulset vminsert -n victoria-metrics
```

⚠ 注意：`kubectl delete statefulset` 不會刪除 PVC（持久存儲），如果沒有手動刪除 PVC，這些存儲資源仍然會保留。

(2) 創建 `Deployment`

📌 新 `Deployment` 配置

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vminsert
  namespace: victoria-metrics
```





```
spec:
  replicas: 3  # 動態擴展
  selector:
    matchLabels:
      app: vminsert
  template:
    metadata:
      labels:
        app: vminsert
    spec:
      containers:
        - name: vminsert
          image: victoriametrics/vminsert:v1.93.2
          args:
            - "--storageNode=vmstorage-0.vmstorage:8482"
            - "--storageNode=vmstorage-1.vmstorage:8482"
          ports:
            - containerPort: 8480
              name: http
          readinessProbe:
            httpGet:
              path: /health
              port: 8480
            initialDelaySeconds: 5
            periodSeconds: 10
          livenessProbe:
            httpGet:
              path: /health
              port: 8480
            initialDelaySeconds: 15
            periodSeconds: 20
        resources:
          requests:
            cpu: "500m"
            memory: "512Mi"
          limits:
            cpu: "1000m"
            memory: "1Gi"
      terminationGracePeriodSeconds: 10
```





◆ 部署 Deployment

```
kubectl apply -f vminsert-deployment.yaml
```

(3) 更新 `Service`

⚠ 關鍵變更：

- `StatefulSet` 之前用的是 **Headless Service** (`clusterIP: None`), 因為 `StatefulSet` 需要固定 DNS 名稱。
- `Deployment` 應該使用 **ClusterIP Service** 來提供單一訪問點，並確保 **負載均衡**。

📌 更新後的 `Service`

```
apiVersion: v1
kind: Service
metadata:
  name: vminsert
  namespace: victoria-metrics
spec:
  ports:
    - port: 8480
      targetPort: 8480
      protocol: TCP
      name: http
  selector:
    app: vminsert
```

◆ 部署 Service

```
kubectl apply -f vminsert-service.yaml
```





5. 測試與驗證

(1) 確認 vminsert 運行

```
kubectl get pods -n victoria-metrics
```

應該看到：

NAME	READY	STATUS	RESTARTS	AGE
vminsert-6bdf9c7f6f-q9w21	1/1	Running	0	10s
vminsert-6bdf9c7f6f-hj7d3	1/1	Running	0	10s
vminsert-6bdf9c7f6f-xw8y9	1/1	Running	0	10s

(2) 測試健康狀態

```
kubectl port-forward svc/vminsert 8480 -n victoria-metrics
curl http://localhost:8480/health
```

如果返回 `200 OK`，表示 `vminsert` 正常運行。

(3) 測試數據寫入

使用 `curl` 測試數據寫入：

```
curl -X POST "http://vminsert.victoria-
metrics.svc.cluster.local:8480/insert/0/prometheus/api/v1/write" \
-d '<data_payload>'
```

6. 影響與 CI/CD Pipeline 更新

(1) CI/CD 變更點

如果公司的 CI/CD pipeline 目前使用 `StatefulSet`，則應將 `StatefulSet` 替換為 `Deployment`，例如：





- **Helm Chart :**

- 若 `values.yaml` 內的 `vminsert` 為 `StatefulSet`，需改為 `Deployment`。

- **ArgoCD / Kustomize :**

- 更新 `kustomization.yaml`，確保 `vminsert` 使用 `Deployment`。

- **GitOps :**

- 確保 `vminsert` 相關的 `Helm` 或 `K8s` 配置使用 `Deployment`。

(2) 滾動更新機制

部署 `Deployment` 之後，每次更新 `vminsert` 版本時，K8s 會自動滾動更新：

```
kubectl set image deployment/vminsert vminsert=victoriametrics/vminsert:v1.94.0  
-n victoria-metrics
```

7. 總結

比較項目	StatefulSet	Deployment	`vminsert` 適合哪個？
擴展性	✗ 需手動配置	✓ 可動態擴展	✓ Deployment
固定 Pod 名稱	✓ 需要	✗ 無需	✗ `vminsert` 不需要
滾動更新	✗ 較困難	✓ 內建支持	✓ Deployment
負載均衡	⚠ 需要手動管理	✓ 透過 Service 自動分配	✓ Deployment

🚀 最終結論

- 應該使用 `Deployment` 來部署 `vminsert`，而非 `StatefulSet`。
- 更新 CI/CD pipeline，確保 `vminsert` 使用 `Deployment`。
- 確保 `Service` 使用 ClusterIP 來負載均衡 `vminsert` 副本。



這樣可以確保 **VictoriaMetrics Cluster** 的 擴展性、可維護性與高可用性！🚀



詳細說明: --storageNode等等的啟動參數要寫在哪裡



Kubernetes 部署 `vminsert`：如何正確設置 `--storageNode` 等啟動參數

1. `--storageNode` 參數的作用

在 **VictoriaMetrics Cluster** 中：

- `vminsert` 是負責接收監控數據並將其寫入 `vmstorage`。
- `--storageNode` 參數用於 **指定 `vmstorage` 節點的地址**，確保 `vminsert` 能夠正確將數據轉發到存儲層。

2. Kubernetes 部署時，`--storageNode` 應該寫在哪裡？

在 K8s **Deployment** 或 **StatefulSet** 部署 `vminsert` 時，`--storageNode` 參數應該寫入：

1. **直接在 `args` 配置中指定** (推薦)。
2. **使用 ConfigMap 存儲**，方便修改。
3. **使用環境變數 `env` 加載**，提高靈活性。





3. `--storageNode` 應該怎麼寫？

方法 1：在 `Deployment` 的 `args` 內直接設定（推薦）

這是最直接、最容易維護的方法。

📌 修改 `vminsert-deployment.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vminsert
  namespace: victoria-metrics
spec:
  replicas: 3
  selector:
    matchLabels:
      app: vminsert
  template:
    metadata:
      labels:
        app: vminsert
    spec:
      containers:
        - name: vminsert
          image: victoriametrics/vminsert:v1.93.2
          args:
            - "--storageNode=vmstorage-0.vmstorage:8482"
            - "--storageNode=vmstorage-1.vmstorage:8482"
      ports:
        - containerPort: 8480
          name: http
      readinessProbe:
        httpGet:
          path: /health
          port: 8480
        initialDelaySeconds: 5
        periodSeconds: 10
      livenessProbe:
```





```
httpGet:  
  path: /health  
  port: 8480  
initialDelaySeconds: 15  
periodSeconds: 20  
resources:  
  requests:  
    cpu: "500m"  
    memory: "512Mi"  
  limits:  
    cpu: "1000m"  
    memory: "1Gi"
```

- ◆ 優點 ✅ 參數一目了然，容易修改。
- ✅ 適合靜態 `vmstorage` 部署（即 `vmstorage-0`，`vmstorage-1` 等不變的情境）。

方法 2：使用 ConfigMap 儲存 `--storageNode`

如果 `vmstorage` 節點列表會變更，或需要統一管理配置，則可以使用 **ConfigMap** 來存儲 `--storageNode` 參數。

📌 建立 `vminsert-configmap.yaml`

```
apiVersion: v1  
kind: ConfigMap  
metadata:  
  name: vminsert-config  
  namespace: victoria-metrics  
data:  
  storage-nodes: "--storageNode=vmstorage-0.vmstorage:8482 --  
storageNode=vmstorage-1.vmstorage:8482"
```

◆ 部署 ConfigMap





```
kubectl apply -f vminsert-configmap.yaml
```

📌 修改 `vminsert-deployment.yaml`，將 `--storageNode` 參數從 ConfigMap 加載

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vminsert
  namespace: victoria-metrics
spec:
  replicas: 3
  selector:
    matchLabels:
      app: vminsert
  template:
    metadata:
      labels:
        app: vminsert
    spec:
      containers:
        - name: vminsert
          image: victoriametrics/vminsert:v1.93.2
          args:
            - "$(STORAGE_NODES)"
          env:
            - name: STORAGE_NODES
              valueFrom:
                configMapKeyRef:
                  name: vminsert-config
                  key: storage-nodes
```

◆ 優點 ✅ 參數集中管理，可透過 `kubectl edit configmap vminsert-config` 修改，不影響 Pod 運行。

✅ 適合動態 `vmstorage` 設定（如果 `vmstorage` 節點會變動）。

✅ 適合 GitOps 流程，便於 Helm 或 Kustomize 管理。





方法 3：使用環境變數 `env` 儲存 `--storageNode`

如果 `--storageNode` 參數較長，可以使用 `env` 來管理，而不是直接寫在 `args`。

📌 修改 `vminsert-deployment.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vminsert
  namespace: victoria-metrics
spec:
  replicas: 3
  selector:
    matchLabels:
      app: vminsert
  template:
    metadata:
      labels:
        app: vminsert
    spec:
      containers:
        - name: vminsert
          image: victoriametrics/vminsert:v1.93.2
          args:
            - "--storageNode=$(STORAGE_NODES)"
      env:
        - name: STORAGE_NODES
          value: "vmstorage-0.vmstorage:8482,vmstorage-1.vmstorage:8482"
```

◆ 優點 ✅ 可以用 CI/CD 管理環境變數（如 ArgoCD, GitHub Actions 等）。

✅ 不需要 ConfigMap，也能透過 `kubectl set env` 即時更新。

4. 哪種方法最適合你的環境？





方法	適用場景	優點	缺點
方法 1：直接寫在 `args`	`vmstorage` 節點固定，簡單環境	易於理解，適合小型集群	變更 `vmstorage` 需更新 Deployment
方法 2：使用 ConfigMap	`vmstorage` 節點可能變更，需集中管理	易於維護，可即時修改	需要多管理 ConfigMap
方法 3：使用環境變數 `env`	配合 CI/CD，`vmstorage` 節點不固定	可動態管理變數，適合自動化	變數長時不易閱讀

◆ 推薦

- 如果 `vmstorage` 節點不會改變 → 方法 1 (`args` 直接寫)
- 如果 `vmstorage` 節點會變更 → 方法 2 (ConfigMap)
- 如果 `vmstorage` 來自動態發現 (如 DNS SRV) → 方法 3 (環境變數)

5. 如何在 CI/CD (Helm / ArgoCD) 中管理 `--storageNode`

(1) 使用 Helm Values

如果公司使用 **Helm** 部署，可以在 `values.yaml` 內設定 `vmstorage` 節點：

```
vminsert:  
  storageNodes:  
    - "vmstorage-0.vmstorage:8482"  
    - "vmstorage-1.vmstorage:8482"
```

然後在 `vminsert-deployment.yaml` 使用 `tpl`：





```
args:  
  - "--storageNode={{ join \" --storageNode=\" .Values.vminsert.storageNodes }}"
```

(2) 使用 ArgoCD Kustomize

在 `kustomization.yaml` 內動態替換 `--storageNode` :

```
configMapGenerator:  
  - name: vminsert-config  
    literals:  
      - storage-nodes="--storageNode=vmstorage-0.vmstorage:8482 --  
        storageNode=vmstorage-1.vmstorage:8482"
```

6. 結論

- `--storageNode` 參數應該放在 `args`，但可用 **ConfigMap** 或 **環境變數** 提高靈活性。
- **K8s `Deployment` 最適合 `vminsert`**，不需要 `StatefulSet`。
- **CI/CD 最佳做法：**
 - `Helm` 使用 `values.yaml`。
 - `ArgoCD` 使用 `ConfigMap` 動態替換。

這樣可以讓 `vminsert` 部署更靈活，擴展性更強！



詳細說明:我被要求vminsert可用的cpu、mem分配要合理，但我查不到建議值





如何為 `vminsert` 設定合理的 CPU 和 Memory 資源

1. 為什麼 `vminsert` 需要合理的 CPU & Memory 設定？

- `vminsert` 是 VictoriaMetrics Cluster 數據寫入入口，它的 資源分配會直接影響數據寫入性能。
- 過少的資源：寫入請求積壓，導致 `vmagent` 或 Prometheus `remoteWrite` 超時。
- 過多的資源：浪費硬體資源，增加運營成本。

🚀 我們的目標：找到一個 合理的 CPU & Memory 分配值，確保 `vminsert` 運行流暢，並有足夠資源應對高負載。

2. `vminsert` CPU & Memory 建議

官方沒有提供明確的 CPU/Memory 建議，但我們可以根據 **工作負載大小** 來設計：

監控數據規 模	建議			適用場景
	寫入流量 (Metric/s)	建議 CPU	Memory	
小規模	1k - 50k metrics/s	`500m - 1 vCPU`	`512Mi - 1Gi`	小型環境 / 測試環境
中等規模	50k - 500k metrics/s	`1 - 2 vCPU`	`1Gi - 2Gi`	正式環境，適用於企業級監控
大規模	500k - 1M+ metrics/s	`2 - 4 vCPU`	`2Gi - 4Gi`	需要高性能、高吞吐
超大規模	1M+ metrics/s	`4 - 8+ vCPU`	`4Gi - 8Gi+`	需要極端高可用，適用於大型 SaaS 監控平台

◆ 一般來說：



- 中等規模 (`50k - 500k metrics/s`): `1 vCPU / 1Gi Memory` 是不錯的起點。
- 需要更高吞吐量時：可以透過 增加 `vminsert` 副本數 (`replicas`），而不是無限增加單個 Pod 的 CPU/Memory。

3. 如何監控 `vminsert` 的資源使用情況

我們可以使用 **Prometheus 指標** 來監控 `vminsert` 的 CPU & Memory 使用率，然後根據實際負載來調整資源。

📌 監控 CPU 負載

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

- 如果 CPU 使用率長期超過 80%，應該增加 CPU 資源或 `vminsert` 副本數。

📌 監控 Memory 使用

```
process_resident_memory_bytes{job="vminsert"}
```

- 如果 `vminsert` 記憶體佔用長期接近 `requests.memory`，代表 **應該增加 Memory 限制**。

📌 監控 HTTP 請求時間

```
histogram_quantile(0.95,  
rate(vm_http_request_duration_seconds_bucket{job="vminsert"}[5m]))
```

- 如果 `95th percentile` 請求時間 > 500ms，表示 寫入過載，可能需要擴容 `vminsert`。





4. Kubernetes `vminsert` CPU & Memory 設定

根據 **監控數據規模**，我們將 `vminsert` 的 `resources.requests` 和 `resources.limits` 設定為合理值。

📌 最佳實踐 (`vminsert-deployment.yaml`)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vminsert
  namespace: victoria-metrics
spec:
  replicas: 3  # 根據需求調整副本數
  selector:
    matchLabels:
      app: vminsert
  template:
    metadata:
      labels:
        app: vminsert
    spec:
      containers:
        - name: vminsert
          image: victoriametrics/vminsert:v1.93.2
          args:
            - "--storageNode=vmstorage-0.vmstorage:8482"
            - "--storageNode=vmstorage-1.vmstorage:8482"
      ports:
        - containerPort: 8480
          name: http
      resources:
        requests:
          cpu: "1"          # 預留 1 vCPU
          memory: "1Gi"    # 預留 1Gi 記憶體
        limits:
          cpu: "2"          # 最多允許 2 vCPU
          memory: "2Gi"    # 最多允許 2Gi 記憶體
```





5. 如何測試 `vminsert` 是否分配合理的 CPU & Memory

1. 部署 `vminsert`

```
kubectl apply -f vminsert-deployment.yaml
```

2. 監控 `vminsert` 資源使用

```
kubectl top pod -n victoria-metrics
```

如果看到：

NAME	CPU(cores)	MEMORY(bytes)
vminsert-6bdf9c7f6f-q9w21	200m	800Mi

- **CPU 200m**：目前使用 0.2 核心，還有足夠餘裕。
- **Memory 800Mi**：還有 200Mi 記憶體可以使用，表示 `1Gi` 設定是合理的。

3. 壓測 `vminsert`

- 如果需要測試 `vminsert` 的極限，可以使用 `ab` 或 `wrk` 進行壓測：

```
ab -n 100000 -c 100 -p sample_data.json -T "application/json"
http://vminsert.victoria-
metrics.svc.cluster.local:8480/insert/0/prometheus/api/v1/write
```

- 如果 `vminsert` 無法處理請求，應該增加 **CPU/Memory** 或 **擴展 replicas**。

6. 如何調整 `vminsert` 副本數來優化資源





- 當 `vminsert` CPU 長期超過 80% → 增加 `replicas`

```
kubectl scale deployment vminsert --replicas=5 -n victoria-metrics
```

- 當 `vminsert` 記憶體長期接近 `limits` → 增加 `memory.requests`

```
resources:  
  requests:  
    memory: "2Gi"  # 從 1Gi 提升至 2Gi  
  limits:  
    memory: "4Gi"  # 限制最多 4Gi
```

7. 結論

如何設定 `vminsert` CPU & Memory ?

- 小規模 (`<50k metrics/s`) : `500m CPU / 512Mi Memory`
- 中等規模 (`50k - 500k metrics/s`) : `1 CPU / 1Gi Memory`
- 大規模 (`>500k metrics/s`) : `2+ CPU / 2+ Gi Memory`
- 超大規模 (`>1M metrics/s`) : `4+ CPU / 4+ Gi Memory`

最佳做法

- 先設定 `1 vCPU / 1Gi Memory`，透過 `kubectl top pod` 監控實際使用量，再調整。
- CPU 過高時，增加 `vminsert` 副本數，而不是單純增加 CPU。
- Memory 過高時，增加 `memory.requests`，但應避免過量配置浪費資源。
- 定期監控 Prometheus 指標，確保 `vminsert` 擁有足夠資源應對寫入流量。

這樣可以確保 **vminsert** 運行流暢，且資源分配合理！🚀





詳細說明:vmagent cluster有3個member，我被要求
vmagent cluster可用的cpu、mem分配要合理，但我查不到
建議值



如何為 `vmagent` Cluster 設定合理的 CPU 和 Memory 分配

1. `vmagent` 在 Cluster 架構中的角色

在 VictoriaMetrics Cluster 架構中：

- `vmagent` 是 Prometheus `remote_write` 接收端，負責從 Prometheus、`node_exporter`、`cAdvisor` 等監控來源抓取數據，然後將數據寫入 `vminsert`。
- `vmagent` 不存儲數據，但它會：
 1. 定期抓取監控數據 (`scrape targets`)。
 2. 執行數據過濾和轉換 (如 `relabel_config`)。
 3. 進行批量傳輸 (`batching`) 以減少 `vminsert` 負擔。

因此，`vmagent` 的 CPU & Memory 需求取決於：

- **Scrape Targets 數量** (抓取多少監控指標？)
- **Scrape 頻率** (每 15s 一次，還是 60s？)
- **Data Retention** (是否需要緩存數據？)
- **並行度** (是否有高併發數據傳輸需求？)





2. `vmagent` CPU & Memory 建議

目前官方沒有明確提供 `vmagent` 資源需求，但我們可以根據 **監控數據規模** 來制定合理的預設值：

監控數據規模	Scrape Targets 數量	Scrape 頻率	建議 CPU	建議 Memory	適用場景
小規模	`1k - 50k metrics/s`	`30s - 60s`	`500m - 1 vCPU`	`512Mi - 1Gi`	測試環境、小型監控
中等規模	`50k - 500k metrics/s`	`15s - 30s`	`1 - 2 vCPU`	`1Gi - 2Gi`	企業級監控
大規模	`500k - 1M+ metrics/s`	`5s - 15s`	`2 - 4 vCPU`	`2Gi - 4Gi`	需要高並發抓取
超大規模	`1M+ metrics/s`	`1s - 5s`	`4 - 8+ vCPU`	`4Gi - 8Gi+`	需要極端高可用

3. 監控 `vmagent` 的 CPU & Memory 使用情況

`vmagent` 本身會暴露 Prometheus 指標，以下是 **如何監控 `vmagent` 資源使用** 的方法：

📌 監控 CPU 負載

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

- 如果 CPU 使用率長期超過 80%，應該 **增加 CPU** 或 **水平擴展**（增加 `replicas`）。

📌 監控 Memory 使用

```
process_resident_memory_bytes{job="vmagent"}
```





- 如果記憶體使用量接近 `requests.memory`，則應該增加 Memory 限制。

📌 監控 Scrape 成功率

```
rate(vm_promscrape_scrapes_total{job="vmagent"}[5m])
```

- 如果抓取數據頻率 低於預期，可能是 `vmagent` 過載。

📌 監控 `remoteWrite` 傳輸情況

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

- 如果 `remoteWrite` 延遲或錯誤率上升，可能是 CPU/Memory 過載，導致數據丟失。

4. Kubernetes `vmagent` CPU & Memory 設定

根據 **監控數據規模**，我們將 `vmagent` 的 `requests` 和 `limits` 設定為合理值。

📌 最佳實踐 (`vmagent-deployment.yaml`)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vmagent
  namespace: victoria-metrics
spec:
  replicas: 3 # 3個vmagent 成員
  selector:
    matchLabels:
      app: vmagent
  template:
    metadata:
      labels:
        app: vmagent
```





```
spec:
  containers:
    - name: vmagent
      image: victoriametrics/vmagent:v1.93.2
      args:
        - "-remoteWrite.url=http://vminsert.victoria-metrics.svc.cluster.local:8480/insert/0/prometheus/api/v1/write"
        - "-promscrape.config=/etc/vmagent/vmagent.yml"
      ports:
        - containerPort: 8429
          name: http
      resources:
        requests:
          cpu: "1"          # 預留 1 vCPU
          memory: "1Gi"    # 預留 1Gi 記憶體
        limits:
          cpu: "2"          # 最多允許 2 vCPU
          memory: "2Gi"    # 最多允許 2Gi 記憶體
```

5. 如何測試 `vmagent` 是否分配合理的 CPU & Memory

(1) 檢查 `vmagent` 實際 CPU & Memory 使用情況

```
kubectl top pod -n victoria-metrics
```

如果看到：

NAME	CPU(cores)	MEMORY(bytes)
vmagent-6bdf9c7f6f-q9w21	700m	900Mi

- **CPU 700m (0.7 核心)**：合理，因為 `requests.cpu` 設定為 1 vCPU。
- **Memory 900Mi**：合理，因為 `requests.memory` 設定為 1Gi。





📌 如果 CPU 使用率長期超過 `requests.cpu`

```
kubectl scale deployment vmagent --replicas=5 -n victoria-metrics
```

📌 如果 Memory 長期接近 `limits.memory`

```
resources:  
  requests:  
    memory: "2Gi" # 從 1Gi 提升至 2Gi  
  limits:  
    memory: "4Gi" # 限制最多 4Gi
```

6. `vmagent` Cluster 的最佳擴展策略

(1) 設置 `replicas` 增強高可用性

如果 **Scrape Targets** 太多，可以 **增加 `replicas`**：

```
kubectl scale deployment vmagent --replicas=5 -n victoria-metrics
```

這樣 `vmagent` 可以分散數據拉取負載，提高效率。

(2) 設置 `HorizontalPodAutoscaler` (HPA)

如果 **監控數據負載會動態變化**，可以使用 HPA 自動擴展：📌 建立 `vmagent-hpa.yaml`

```
apiVersion: autoscaling/v2  
kind: HorizontalPodAutoscaler  
metadata:  
  name: vmagent-hpa  
  namespace: victoria-metrics  
spec:
```





```
scaleTargetRef:  
  apiVersion: apps/v1  
  kind: Deployment  
  name: vmagent  
minReplicas: 3  
maxReplicas: 10  
metrics:  
  - type: Resource  
    resource:  
      name: cpu  
      target:  
        type: Utilization  
        averageUtilization: 75 # CPU 達到 75% 時擴展  
  - type: Resource  
    resource:  
      name: memory  
      target:  
        type: Utilization  
        averageUtilization: 80 # 記憶體達到 80% 時擴展
```

◆ 部署 HPA

```
kubectl apply -f vmagent-hpa.yaml
```

7. 結論

- ✓ 小型集群 (`<50k metrics/s`) → `500m CPU / 512Mi Memory`
- ✓ 中等集群 (`50k - 500k metrics/s`) → `1 vCPU / 1Gi Memory`
- ✓ 大型集群 (`>500k metrics/s`) → `2+ vCPU / 2+ Gi Memory`
- ✓ 超大規模 (`>1M metrics/s`) → `4+ vCPU / 4+ Gi Memory`

🚀 最佳做法





1. 使用 `kubectl top pod` 監控資源消耗，根據實際數據調整 `requests.cpu` 和 `requests.memory`。
2. 使用 `replicas` 或 HPA 來擴展 `vmagent`，而不是無限增加單 Pod 的資源。
3. 監控 `vmagent` Prometheus 指標，確保數據抓取不會延遲或丟失。



公司有用gitlab pipeline，用statefulset把vminsert佈到k8s，要求我改為用Deployment，但我對這statefulset、Deployment都不熟，你能教我嗎



1. 什麼是 StatefulSet 與 Deployment？

在 **Kubernetes (K8s)** 中，`StatefulSet` 和 `Deployment` 是兩種不同的 **工作負載管理器 (Workload Controllers)**，它們負責管理應用程式的部署、擴展和滾動更新。

特性	StatefulSet	Deployment
適用場景	需要固定名稱、存儲的應用	無狀態應用
Pod 名稱	`pod-0`、`pod-1`（固定名稱）	`pod-randomid`（隨機名稱）
DNS 名稱	每個 Pod 都有獨立 DNS（適用於資料庫）	所有 Pod 共享 Service DNS
滾動更新	有序滾動更新（適合有狀態應用）	可快速更新
擴展方式	需確保新 Pod 連接資料存儲	可直接新增 Pod
持久存儲	PVC 綁定特定 Pod	無需 PVC

簡單來說

- **StatefulSet** 適合 需要 固定 DNS 名稱、持久存儲、有序啟動/關閉 的應用，例如 **資料庫**、**ZooKeeper**、**Elasticsearch**。





- **Deployment** 適合 無狀態 (Stateless) 的應用，例如 **API 服務、Web 應用、負載均衡器**。

🚀 `vminsert` 是無狀態應用，因此應該使用 `Deployment`，而不是 `StatefulSet`。

2. 為什麼 `vminsert` 應該用 **Deployment** 而非 **StatefulSet**？

🚫 不需要 StatefulSet 的原因

1. `vminsert` 無需固定 Pod 名稱

- `StatefulSet` 會分配固定的 Pod 名稱（如 `vminsert-0`，`vminsert-1`），但 `vminsert` 不需要這些固定名稱，因為它只負責將數據寫入 `vmstorage`。

2. `vminsert` 不需要持久存儲

- `StatefulSet` 會為每個 Pod 建立獨立的 PVC (Persistent Volume Claim)，但 `vminsert` 不存儲任何數據，所以不需要 PVC。

3. `vminsert` 需要動態擴展

- `Deployment` 支持更靈活的 **擴展** (`scale`) 和**滾動更新** (`rolling update`)，適合需要彈性部署的服務。

✅ **結論：** 應該將 `vminsert` 從 `StatefulSet` 轉換為 `Deployment`，以提高擴展性、滾動更新速度，並簡化管理。

3. 如何將 GitLab Pipeline 內的 StatefulSet 轉換為 Deployment？





(1) 原始 StatefulSet 配置

你可能的 `StatefulSet` 配置類似這樣：

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: vminsert
  namespace: victoria-metrics
spec:
  serviceName: "vminsert"
  replicas: 2
  selector:
    matchLabels:
      app: vminsert
  template:
    metadata:
      labels:
        app: vminsert
    spec:
      containers:
        - name: vminsert
          image: victoriametrics/vminsert:v1.93.2
          args:
            - "--storageNode=vmstorage-0.vmstorage:8482"
            - "--storageNode=vmstorage-1.vmstorage:8482"
          ports:
            - containerPort: 8480
              name: http
```

(2) 修改為 Deployment

我們將 `StatefulSet` 轉換為 `Deployment`，主要修改：

1. 將 `StatefulSet` 改為 `Deployment`。
2. 移除 `serviceName` (`StatefulSet` 需要， `Deployment` 不需要)。



3. **Pod** 名稱不再固定，由 K8s 自動分配。
4. 支援滾動更新（可快速更新 `vminsert`）。

📌 修改後的 `vminsert-deployment.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vminsert
  namespace: victoria-metrics
spec:
  replicas: 3  # 可以靈活擴展
  selector:
    matchLabels:
      app: vminsert
  template:
    metadata:
      labels:
        app: vminsert
    spec:
      containers:
        - name: vminsert
          image: victoriametrics/vminsert:v1.93.2
          args:
            - "--storageNode=vmstorage-0.vmstorage:8482"
            - "--storageNode=vmstorage-1.vmstorage:8482"
          ports:
            - containerPort: 8480
              name: http
          resources:
            requests:
              cpu: "1"
              memory: "1Gi"
            limits:
              cpu: "2"
              memory: "2Gi"
```

◆ 部署 Deployment



```
kubectl apply -f vminsert-deployment.yaml
```

(3) 更新 GitLab Pipeline

在 `*.gitlab-ci.yml` 內，修改 K8s 部署步驟：📌 修改前

```
deploy:  
  script:  
    - kubectl apply -f vminsert-statefulset.yaml
```

📌 修改後

```
deploy:  
  script:  
    - kubectl apply -f vmissert-deployment.yaml
```

🚀 這樣 GitLab CI/CD 會自動使用 `Deployment` 來部署 `vminsert` !

4. 測試與驗證

(1) 確認 `vminsert` 運行

```
kubectl get pods -n victoria-metrics
```

應該看到：

NAME	READY	STATUS	RESTARTS	AGE
vminsert-6bdf9c7f6f-q9w21	1/1	Running	0	10s





vminsert-6bdf9c7f6f-hm2v8	1/1	Running	0	10s
vminsert-6bdf9c7f6f-xw8y9	1/1	Running	0	10s

(2) 測試 `vminsert` 健康狀態

```
kubectl port-forward svc/vminsert 8480 -n victoria-metrics  
curl http://localhost:8480/health
```

如果返回 `200 OK`，表示 `vminsert` 正常運行。

(3) 測試數據寫入

```
curl -X POST "http://vminsert.victoria-  
metrics.svc.cluster.local:8480/insert/0/prometheus/api/v1/write" \  
-d '<data_payload>'
```

5. 結論

✓ `vminsert` 應該使用 `Deployment`，不需要 `StatefulSet`。

✓ 轉換過程

1. 修改 YAML 配置

- `StatefulSet` → `Deployment`
- 刪除 `serviceName`
- 支援滾動更新與自動擴展

2. 修改 GitLab CI/CD Pipeline

- `gitlab-ci.yml` 部署改為 `Deployment`

3. 測試 `vminsert` 是否正常運行

- 使用 `kubectl get pods` 確認運行
- `curl` 測試數據寫入與健康狀態





🚀 這樣可以確保 `vminsert` 更靈活、更容易管理，並符合最佳 Kubernetes 部署實踐！ 🚀



以繁體中文詳細說明:我可以把statefulset裡的
resorceversion、uid、podmanagmentpolicy、
revisionhistorylimit、servicename、
creationtimestamp、apiversion、
terminationmessagepath、terminationmessagepolicy、
volumemounts、restartpolicy、schedulername、
sercuritycontentext、terminationgraceperiodseconds、
updatestrategy、voluneclaimtemplates、status拿掉嗎



在 **Kubernetes** 中，當你將 **StatefulSet** 轉換為 **Deployment**，可以適當地 **刪除或修改某些欄位**。以下是 **你可以刪除的欄位**，以及 **是否影響運行** 的詳細說明。

1. 可以刪除的欄位

以下是 **StatefulSet** 轉為 **Deployment** 時 **可以刪除的欄位**，因為這些欄位對 `Deployment` 來說 **不是必要的** 或者 **沒有影響**。

欄位	是否可刪 除？	原因
`resourceVersion`	<input checked="" type="checkbox"/> 可刪 除	由 Kubernetes 自動管理，Deployment 不需要手動指定。
`uid`	<input checked="" type="checkbox"/> 可刪 除	由 Kubernetes 自動生成，不能手動指定。
`creationTimestamp`	<input checked="" type="checkbox"/> 可刪 除	由 Kubernetes 自動管理，Deployment 無需手動指定。





欄位	是否可刪除？	原因
`apiVersion`	🚫 不能刪除	這是 必要的 ，必須指定 `apps/v1` 來識別 API 版本。
`serviceName`	✅ 可刪除	`Deployment` 不需要 `serviceName`，因為 Pod 透過 `Service` 來負載均衡。
`podManagementPolicy`	✅ 可刪除	只適用於 `StatefulSet`，控制 Pod 啟動順序，Deployment 不需要。
`revisionHistoryLimit`	✅ 可刪除	`StatefulSet` 和 `Deployment` 都可以使用，但 Deployment 默認會保留 `10` 個版本。
`terminationMessagePath`	✅ 可刪除	預設為 `/dev/termination-log`，可以不手動指定。
`terminationMessagePolicy`	✅ 可刪除	預設為 `File`，不手動指定也沒影響。
`restartPolicy`	🚫 不能刪除	必須存在 ，且 Deployment 預設為 `Always`，所以應確保 `restartPolicy: Always`。
`schedulerName`	✅ 可刪除	預設為 `default-scheduler`，如果你沒有特別的調度需求，則可以刪除。
`securityContext`	✗ 不建議刪除	若有安全性需求（如 `runAsUser`），則應該保留，否則可以刪除。
`terminationGracePeriodSeconds`	✅ 可刪除	預設為 `30` 秒，不手動指定也可以。
`updateStrategy`	✅ 可刪除	`StatefulSet` 用來定義滾動更新方式，而 `Deployment` 預設使用 `RollingUpdate`，因此可以刪除。
`volumeClaimTemplates`	🚫 不能刪除	如果你的 `StatefulSet` 有 PVC 持久化需求，則不能刪除，否則可以刪除。
`status`	✅ 可刪除	由 Kubernetes 動態生成，Deployment 不需要手動管理 `status` 欄位。





2. 轉換範例：刪除不需要欄位

(1) 原本的 `StatefulSet`

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: vminsert
  namespace: victoria-metrics
  resourceVersion: "12345"
  uid: "67890"
  creationTimestamp: "2024-02-26T12:34:56Z"
spec:
  serviceName: "vminsert"
  replicas: 2
  podManagementPolicy: OrderedReady
  revisionHistoryLimit: 10
  updateStrategy:
    type: RollingUpdate
  volumeClaimTemplates:
    - metadata:
        name: data
      spec:
        accessModes: ["ReadWriteOnce"]
        resources:
          requests:
            storage: 10Gi
    template:
      metadata:
        labels:
          app: vminsert
      spec:
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        restartPolicy: Always
        schedulerName: default-scheduler
        securityContext:
```





```
    runAsUser: 1000
    terminationGracePeriodSeconds: 30
status:
  replicas: 2
  readyReplicas: 2
```

(2) 刪除不需要欄位後的 `Deployment`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vminsert
  namespace: victoria-metrics
spec:
  replicas: 3 # 可以動態擴展
  selector:
    matchLabels:
      app: vminsert
  template:
    metadata:
      labels:
        app: vminsert
    spec:
      containers:
        - name: vminsert
          image: victoriametrics/vminsert:v1.93.2
          args:
            - "--storageNode=vmstorage-0.vmstorage:8482"
            - "--storageNode=vmstorage-1.vmstorage:8482"
          ports:
            - containerPort: 8480
              name: http
  restartPolicy: Always
```





3. 轉換過程的步驟

(1) 刪除 `StatefulSet`

如果你的 `StatefulSet` 沒有 PVC (持久存儲)，可以直接刪除：

```
kubectl delete statefulset vminsert -n victoria-metrics
```

⚠️ 如果有 PVC (VolumeClaimTemplates)，建議先保留 PVC，避免資料遺失。

(2) 部署 `Deployment`

```
kubectl apply -f vminsert-deployment.yaml
```

(3) 確認 `vminsert` 運行

```
kubectl get pods -n victoria-metrics
```

應該會看到：

NAME	READY	STATUS	RESTARTS	AGE
vminsert-6bdf9c7f6f-q9w2l	1/1	Running	0	10s
vminsert-6bdf9c7f6f-hm2v8	1/1	Running	0	10s
vminsert-6bdf9c7f6f-xw8y9	1/1	Running	0	10s

(4) 測試 `vminsert`

```
kubectl port-forward svc/vminsert 8480 -n victoria-metrics
curl http://localhost:8480/health
```

如果返回 `200 OK`，表示 `vminsert` 正常運行。





4. 總結

✓ 可以刪除的欄位

- `resourceVersion`
- `uid`
- `creationTimestamp`
- `serviceName`
- `podManagementPolicy`
- `revisionHistoryLimit`
- `terminationMessagePath`
- `terminationMessagePolicy`
- `schedulerName`
- `terminationGracePeriodSeconds`
- `updateStrategy`
- `status`

🚫 不能刪除的欄位

- `apiVersion` (Deployment 必須使用 `apps/v1`)
- `restartPolicy` (Deployment 必須設為 `Always`)
- `volumeClaimTemplates` (如果 `vminsert` 需要持久存儲)

這樣的轉換確保 **vminsert** 正常運行、資源配置合理，並提升擴展性與滾動更新效率！🚀



以繁體中文詳細說明:我可以把statefulset裡的metadata、dnsconfig、





在 Kubernetes **StatefulSet** 轉換為 **Deployment** 時，你可以 **刪除或修改某些欄位**，但某些欄位仍然是必要的。以下是 **你問的 `metadata` 和 `dnsConfig` 是否可以刪除的詳細解釋**。

1. `metadata` 和 `dnsConfig` 可不可以刪除？

欄位	是否可以刪除？	說明
`metadata`	🚫 不能刪除	`metadata` 是 Kubernetes 物件的基本資訊，至少需要 `name`、`namespace`。
`dnsConfig`	✅ 可以刪除	如果沒有特別的 DNS 設定，則可以刪除， Kubernetes 會使用預設的 Cluster DNS 。

2. `metadata` 是否能刪除？

🚫 `metadata` 不能刪除，但可以簡化。

StatefulSet 的 `metadata` (原始設定)

```
metadata:  
  name: vminsert  
  namespace: victoria-metrics  
  labels:  
    app: vminsert  
  annotations:  
    example.com/managed-by: "gitlab"  
  resourceVersion: "12345"
```





```
uid: "67890"  
creationTimestamp: "2024-02-26T12:34:56Z"
```

Deployment 版本 (簡化)

- ✓ 你可以刪除 `resourceVersion`、`uid` 和 `creationTimestamp`，但 `name` 和 `namespace` 不能刪除：

```
metadata:  
  name: vminsert  
  namespace: victoria-metrics  
  labels:  
    app: vminsert
```

3. `dnsConfig` 是否能刪除？

- ✓ 如果 StatefulSet 沒有特別的 DNS 設定，可以刪除 `dnsConfig`，Kubernetes 預設會使用 Cluster DNS。

StatefulSet 原始 `dnsConfig` 設定

```
dnsConfig:  
  nameservers:  
    - 1.1.1.1  
  searches:  
    - example.com
```

這代表 Pod 會優先使用 `1.1.1.1` 這個 DNS 伺服器，並且搜尋 `example.com`。

Deployment 版本

- 🚀 如果不需要特別的 DNS 設定，可以刪除 `dnsConfig`：



```
# Deployment 沒有 dnsConfig，使用 Kubernetes 預設 Cluster DNS
```

📌 但如果你需要保留 `dnsConfig`，可以這樣寫：

```
dnsConfig:  
  options:  
    - name: ndots  
      value: "5"
```

這可以 **控制 DNS 解析行為**，但一般情況下 **Kubernetes 預設的 DNS 設定已經足夠**。

4. 刪除 `metadata` & `dnsConfig` 後的 Deployment

(1) 原本的 StatefulSet

```
apiVersion: apps/v1  
kind: StatefulSet  
metadata:  
  name: vminsert  
  namespace: victoria-metrics  
  labels:  
    app: vminsert  
  annotations:  
    example.com/managed-by: "gitlab"  
  resourceVersion: "12345"  
  uid: "67890"  
  creationTimestamp: "2024-02-26T12:34:56Z"  
spec:  
  serviceName: "vminsert"  
  replicas: 2  
  template:  
    metadata:  
      labels:  
        app: vminsert
```





```
spec:  
  dnsConfig:  
    nameservers:  
      - 1.1.1.1  
    searches:  
      - example.com  
  containers:  
    - name: vminsert  
      image: victoriametrics/vminsert:v1.93.2
```

(2) 轉換後的 Deployment

🚀 刪除 `dnsConfig`，簡化 `metadata`，轉換為 `Deployment`

```
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: vminsert  
  namespace: victoria-metrics  
  labels:  
    app: vminsert  
spec:  
  replicas: 3 # 可擴展的副本數  
  selector:  
    matchLabels:  
      app: vminsert  
  template:  
    metadata:  
      labels:  
        app: vminsert  
    spec:  
      containers:  
        - name: vminsert  
          image: victoriametrics/vminsert:v1.93.2
```





5. 轉換與測試

(1) 刪除 StatefulSet

如果你已經部署了 `StatefulSet`，需要先刪除它：

```
kubectl delete statefulset vminsert -n victoria-metrics
```

(2) 部署新的 `Deployment`

```
kubectl apply -f vminsert-deployment.yaml
```

(3) 確認 `vminsert` 運行

```
kubectl get pods -n victoria-metrics
```

應該看到：

NAME	READY	STATUS	RESTARTS	AGE
vminsert-6bdf9c7f6f-q9w21	1/1	Running	0	10s
vminsert-6bdf9c7f6f-hm2v8	1/1	Running	0	10s
vminsert-6bdf9c7f6f-xw8y9	1/1	Running	0	10s

(4) 測試 `vminsert`

```
kubectl port-forward svc/vminsert 8480 -n victoria-metrics
curl http://localhost:8480/health
```

如果返回 `200 OK`，表示 `vminsert` 正常運行。





6. 總結

欄位	可以刪除？	說明
`metadata`	🚫 不能完全 刪除	但可以刪除 `resourceVersion`、`uid` 和 `creationTimestamp`，只保留 `name` 和 `namespace`。
`dnsConfig`	✅ 可以刪除	如果沒有特殊 DNS 需求，可以刪除，Kubernetes 會使用預設的 Cluster DNS。

🚀 最佳做法

1. 保留 `metadata` 的 `name` 和 `namespace`，刪除 `resourceVersion`、`uid`、`creationTimestamp`。
2. 如果 `dnsConfig` 沒有特殊設定，可以刪除，使用 **Kubernetes** 預設 DNS 配置。
3. 轉換為 `Deployment`，確保 `vminsert` 運行穩定，並測試健康狀態。

這樣可以讓 `vminsert` 更靈活、更易管理，符合 **Kubernetes** 的最佳實踐！🎯



不要的字段可以註解還是只能拿掉



在 **Kubernetes YAML** 配置中，不要的字段可以直接刪除，也可以使用註解，但實際上 Kubernetes 不會解析註解，所以 註解的欄位不會被 K8s 識別或影響。

1. 哪些字段可以註解？

- 如果你只是想暫時停用某些設定，可以用 `#` 註解。





- 如果這些欄位在 `StatefulSet` 轉換為 `Deployment` 時不適用，應該直接刪除。

✓ 適合註解的情況

- 你不確定某個欄位是否還需要，可以先註解再測試。
- 你希望保留某些資訊，以便將來可能使用。

✗ 應該直接刪除的情況

- Kubernetes 不會解析註解的欄位，如果你確認某個欄位不適用 Deployment，直接刪除比較乾淨。
- `kubectl apply` 不會處理註解的內容，即使你註解某個欄位，K8s 也不會知道它的存在。

2. 註解的正確方式

(1) 直接刪除不適用欄位 (最佳方式)

這樣可以 確保 Kubernetes 配置文件更簡潔，不會因為冗餘欄位影響部署：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vminsert
  namespace: victoria-metrics
spec:
  replicas: 3
  selector:
    matchLabels:
      app: vminsert
  template:
    metadata:
      labels:
        app: vminsert
    spec:
      containers:
```





```
- name: vminsert
  image: victoriametrics/vminsert:v1.93.2
```

📌 這是最佳做法，因為不適用 `Deployment` 的欄位完全刪除，避免混淆。

(2) 使用註解 (適合暫時保留資訊)

如果你 不確定是否需要刪除某個欄位，可以先註解掉，這樣你可以 隨時恢復：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vminsert
  namespace: victoria-metrics
  # resourceVersion: "12345" # Kubernetes 會自動生成，Deployment 不需要
  # uid: "67890" # 由 K8s 自動分配
  # creationTimestamp: "2024-02-26T12:34:56Z" # 不需要手動設定
spec:
  replicas: 3
  selector:
    matchLabels:
      app: vminsert
  template:
    metadata:
      labels:
        app: vminsert
    spec:
      containers:
        - name: vminsert
          image: victoriametrics/vminsert:v1.93.2
      # dnsConfig:
      #   nameservers:
      #     - 1.1.1.1 # 如果需要自訂 DNS，可以啟用
      #   searches:
      #     - example.com # 預設會使用 Kubernetes 內建 DNS，不一定需要
```





📌 這樣做的好處是，如果將來需要 `dnsConfig`，只要取消註解就能立即使用！

3. 註解 VS 直接刪除：如何選擇？

情境	適合註解 `#`	應該直接刪除
你不確定某個欄位是否必要	<input checked="" type="checkbox"/> 先註解，之後測試是否影響	<input checked="" type="checkbox"/> ✗
這個欄位 K8s 會自動生成（如 `resourceVersion`、`uid`、`status`）		<input checked="" type="checkbox"/> 應該刪除
這個欄位 `Deployment` 不支援（如 `serviceName`、`podManagementPolicy`）	<input checked="" type="checkbox"/> ✗	<input checked="" type="checkbox"/> 應該刪除
這個欄位將來可能會用到（如 `dnsConfig`）	<input checked="" type="checkbox"/> 可以註解	<input checked="" type="checkbox"/> ✗
想讓 YAML 文件更乾淨	<input checked="" type="checkbox"/> ✗	<input checked="" type="checkbox"/> 直接刪除

4. 轉換 `StatefulSet` 時，哪些應該註解？哪些應該刪除？

欄位	應該刪除 (✗)	可以註解 (✓)	原因
`resourceVersion`	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> ✗	由 Kubernetes 自動管理，不需要手動指定。
`uid`	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> ✗	由 Kubernetes 自動生成，無需保留。
`creationTimestamp`	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> ✗	由 Kubernetes 自動填充，不需要手動設定。





欄位	應該刪除 (`X`)	可以註解 (`✓`)	原因
`serviceName`	✓	X	`StatefulSet` 專用，Deployment 不需要。
`podManagementPolicy`	✓	X	`StatefulSet` 控制 Pod 啟動順序，Deployment 不支援。
`revisionHistoryLimit`	X	✓	Deployment 會保留 `10` 個版本，若有特別需求可以啟用。
`terminationMessagePath`	X	✓	預設為 `/dev/termination-log`，可保留但不影響運行。
`terminationMessagePolicy`	X	✓	預設為 `File`，可以註解但不影響運行。
`restartPolicy`	X	✓	Deployment 預設為 `Always`，應該保留。
`schedulerName`	X	✓	如果沒有特殊調度需求，則可註解掉。
`securityContext`	X	✓	如果有 `runAsUser`，應該保留，否則可註解。
`terminationGracePeriodSeconds`	X	✓	預設為 `30` 秒，若不需要可註解。
`updateStrategy`	✓	X	`StatefulSet` 需要，但 `Deployment` 默認 `RollingUpdate`，不需要保留。
`volumeClaimTemplates`	✓	X	如果 `vminsert` 需要持久存儲，應該轉換成 `PersistentVolumeClaim`。
`status`	✓	X	由 Kubernetes 動態生成，應該刪除。
`dnsConfig`	X	✓	如果不需要自訂 DNS，可以註解，但若未來有需求可以保留。

5. 結論





✓ 最佳做法

1. 不適用於 `Deployment` 的欄位（如 `serviceName`，`updateStrategy`），應該直接刪除。
2. **Kubernetes** 會自動管理的欄位（如 `resourceVersion`，`uid`，`status`），應該直接刪除。
3. 將來可能會用到的欄位（如 `dnsConfig`，`terminationMessagePath`），可以註解，而不是刪除。
4. 如果不確定某個欄位是否必要，先註解，測試後再決定是否刪除。

🚀 實際應用

- 如果你希望讓 YAML 文件更簡潔，直接刪除。
- 如果你想保留一些欄位作為未來參考，則可以註解。

這樣可以確保 **Kubernetes** 物件配置最佳化、運行高效且易於維護！🚀



以繁體中文詳細說明:我改成deployment後，argocd的ui的deployment資訊顯示”OutOfSync (This resource is not present in the application's source. It will be deleted from Kubernetes if the prune option is enabled during sync.)”



ArgoCD UI 顯示 `OutOfSync` 並警告 `This resource is not present in the application's source`

當你將 `vminsert` 的 `StatefulSet` 改成 `Deployment`，但 ArgoCD UI 顯示 `outofSync` 並提示 `This resource is not present in the application's source`，這表示 ArgoCD 的 Git Repo 內沒有記錄這個 `Deployment`，但 Kubernetes 內卻有這個資源。

如果 `prune`（修剪）選項啟用的話，ArgoCD 會自動刪除 這些不在 Git 內的資源。





1. 為什麼 ArgoCD 會顯示 `OutOfSync`？

這個錯誤通常有 **三個可能原因**：

(1) Git Repo 內仍然是 `StatefulSet`，但 Kubernetes 內是 `Deployment`

- 你已經 **手動套用** (`kubectl apply -f`) `Deployment`，但 **Git** 內仍然是舊的 `StatefulSet`，ArgoCD 發現兩者不同步，所以標記為 `OutOfSync`。
- 解決方案**：更新 **Git Repo**，讓 **Git** 內的 **YAML** 與 **K8s** 一致。

(2) Git Repo 內的 `StatefulSet` 被刪除，但 ArgoCD 還記得它

- 你在 **Git** 內 **刪除了** `StatefulSet`，但 **Kubernetes** 內的 `StatefulSet` **還存在**，ArgoCD **不知道應該刪除**它，所以標記為 `OutOfSync`。
- 解決方案**：執行 `argocd app sync <app-name>`，讓 **ArgoCD** 強制同步。

(3) ArgoCD 沒有檢測到新的 `Deployment`

- 你已經 **在 Git** 內提交了 `Deployment`，但 **ArgoCD** **沒有成功偵測到新的 Deployment**，可能是因為 ArgoCD 還沒重新同步 (`sync`)。
- 解決方案**：在 **ArgoCD UI** 上點擊 `Sync`，或手動執行 `argocd app sync <app-name>`。

2. 如何解決 `OutOfSync` 問題

✓ 步驟 1：確認 Git Repo 內 YAML 是否正確

如果 **你還沒有把 `StatefulSet` 改成 `Deployment`**，請 **修改 Git 內的 YAML**，並提交變更。

✗ 舊的 `StatefulSet`（刪除它）





```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: vminsert
  namespace: victoria-metrics
spec:
  serviceName: "vminsert"
  replicas: 2
  template:
    metadata:
      labels:
        app: vminsert
    spec:
      containers:
        - name: vminsert
          image: victoriametrics/vminsert:v1.93.2
```

📌 新的 `Deployment` (新增它)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vminsert
  namespace: victoria-metrics
spec:
  replicas: 3
  selector:
    matchLabels:
      app: vminsert
  template:
    metadata:
      labels:
        app: vminsert
    spec:
      containers:
        - name: vminsert
          image: victoriametrics/vminsert:v1.93.2
```





◆ 提交 Git 變更

```
git add .
git commit -m "改為 Deployment 部署 vminsert"
git push origin main
```

✓ 步驟 2：強制 ArgoCD 同步

當你更新了 Git Repo，ArgoCD 通常會自動同步，但有時候 仍然顯示 `outOfSync`，這時候可以手動同步：

```
argocd app sync <app-name>
```

- `<app-name>` 是你的 ArgoCD 應用名稱，例如 `victoriametrics-cluster`。
- 這樣 ArgoCD 會拉取 Git 內的新 YAML 並套用到 Kubernetes。

如果 ArgoCD UI 仍然顯示 `outOfSync`，可以使用：

```
argocd app get <app-name>
```

這樣可以檢查 有哪些資源仍然不同步。

✓ 步驟 3：刪除舊的 `StatefulSet`

如果你在 Git Repo 內刪除了 `StatefulSet`，但 Kubernetes 內還有 `StatefulSet`，ArgoCD 會顯示 `outOfSync`。

📌 先確認 Kubernetes 內還有沒有 `StatefulSet`





```
kubectl get statefulset -n victoria-metrics
```

如果仍然存在 `vminsert` 的 `StatefulSet`，手動刪除：

```
kubectl delete statefulset vminsert -n victoria-metrics
```

這樣 **Kubernetes** 內的 `StatefulSet` 會被刪除，並讓 `Deployment` 正確接管。

✓ 步驟 4：重新同步 ArgoCD

手動執行 `argocd app sync`，確保 `Deployment` 被正確部署

```
argocd app sync <app-name>
```

如果你的 ArgoCD 設定支援 **自動同步** (`auto-sync`)，可以確保 `Deployment` 會在每次 Git 變更時自動更新：

```
syncPolicy:  
  automated:  
    prune: true # 移除 Git 內不存在的資源  
    selfHeal: true # 自動修復不同步的資源
```

這樣 ArgoCD 發現 `StatefulSet` 在 Git 內已刪除，就會自動刪除 **Kubernetes** 內的 `StatefulSet`，確保 `Deployment` 被正確部署。

3. 最終檢查

✓ 確保 `Deployment` 正確執行





```
kubectl get deployment -n victoria-metrics
```

應該看到：

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
vminsert	3/3	3	3	10m

✓ 確保 `StatefulSet` 已刪除

```
kubectl get statefulset -n victoria-metrics
```

如果已經刪除，應該不會有輸出。

✓ 檢查 ArgoCD UI

- `Sync Status` → 應該變成 `Synced` ✓
- `Health Status` → 應該變成 `Healthy` ✓

4. 總結

🚀 ArgoCD UI 顯示 `OutOfSync`，並提示 `This resource is not present in the application's source`，解決方法如下：

1. 檢查 Git Repo，確保 `StatefulSet` 改為 `Deployment`，並提交變更 (`git push`)。
2. 手動同步 ArgoCD (`argocd app sync <app-name>`)，讓 ArgoCD 拉取最新 YAML。
3. 確認 Kubernetes 內沒有 `StatefulSet` (`kubectl get statefulset`)，如有，手動刪除 (`kubectl delete statefulset vminsert -n victoria-metrics`)。
4. 啟用 `prune: true` 讓 ArgoCD 自動刪除 Git 內已刪除的資源，避免未來再次出現 `OutOfSync`。





5. 確認 ArgoCD UI 顯示 `Synced`，並檢查 `vminsert` 是否正常運行 (`kubectl get deployment`)。

這樣 可以確保 `Deployment` 正常取代 `StatefulSet`，不會再出現 `outOfSync` 問題！🎯



以繁體中文詳細說明:(2) Git Repo 內的 StatefulSet 被刪除，但 ArgoCD 還記得它、(3) ArgoCD 沒有檢測到新的 Deployment



ArgoCD `OutOfSync` 問題詳細解析

當 你從 `StatefulSet` 轉換成 `Deployment`，但 ArgoCD UI 仍然顯示 `OutOfSync`，可能是因為以下兩個原因：

(2) Git Repo 內的 StatefulSet 被刪除，但 ArgoCD 還記得它

為什麼 ArgoCD 還記得 `StatefulSet`？

- 當 ArgoCD 之前管理了一個 `StatefulSet`，但你手動刪除了它的 YAML，ArgoCD 不會自動刪除 Kubernetes 內的 `StatefulSet`，除非 `prune`（修剪）選項啟用。
- 這導致 ArgoCD 仍然顯示 `StatefulSet` 的狀態，但它其實已經不在 Git Repo 內了，所以 ArgoCD UI 顯示：

OutOfSync (This resource **is not** present **in** the application's source.
It will be deleted **from** Kubernetes **if** the prune **option is** enabled during sync.)





解決方法

✓ 步驟 1：確認 `StatefulSet` 是否仍然存在於 Kubernetes

執行以下指令，檢查 `StatefulSet` 是否仍然存在：

```
kubectl get statefulset -n victoria-metrics
```

如果輸出如下：

NAME	READY	AGE
vminsert	2/2	10d

這代表 Kubernetes 內仍然有 `StatefulSet`，但它已經不在 Git Repo 內了。

✓ 步驟 2：手動刪除 `StatefulSet`

因為 Git 內已經沒有 `StatefulSet`，所以應該手動刪除 Kubernetes 內的 StatefulSet：

```
kubectl delete statefulset vminsert -n victoria-metrics
```

這樣 Kubernetes 內的 `StatefulSet` 會被刪除，ArgoCD 也不會再標記它為 `OutofSync`。

✓ 步驟 3：同步 ArgoCD

刪除 `StatefulSet` 之後，執行以下指令強制 ArgoCD 重新同步：





```
argocd app sync <app-name>
```

- `<app-name>` 是你的 ArgoCD 應用名稱，例如 `victoriametrics-cluster`。
- 這樣 ArgoCD 會確認 `StatefulSet` 已被刪除，並確保新的 `Deployment` 正確部署。

✓ 步驟 4：啟用 `prune`（可選）

如果你希望 以後當 Git 內刪除某個資源時，ArgoCD 也會自動刪除 Kubernetes 內的對應資源，可以在 ArgoCD 設定 `prune: true`：

```
syncPolicy:  
  automated:  
    prune: true # 移除 Git 內不存在的資源  
    selfHeal: true # 自動修復不同步的資源
```

這樣 當你在 Git 內刪除 `StatefulSet`，ArgoCD 會自動刪除 Kubernetes 內的 `StatefulSet`，確保狀態一致。

(3) ArgoCD 沒有檢測到新的 `Deployment`

為什麼 ArgoCD 沒有檢測到 `Deployment`？

有幾個可能的原因：

1. ArgoCD 還沒執行同步 (`sync`)
 - ArgoCD 不會即時監測 Git Repo 變更，它需要手動同步或設置 `auto-sync`。
2. Git 內的 `Deployment` 有錯誤，ArgoCD 無法解析





- 例如 YAML 語法錯誤，或者 `apiVersion` 寫錯，導致 ArgoCD 無法成功解析 `Deployment`。

3. ArgoCD 應用的 `source` 設定沒有包含 `Deployment`

- 確認 `ArgoCD` 的 `application` 配置，確保它監測的是正確的 Git 路徑。

解決方法

✓ 步驟 1：確認 `Deployment` 是否已經提交到 Git

檢查 Git 內是否有 `Deployment` 配置：

```
git ls-files | grep vminsert-deployment.yaml
```

如果 Git 內還沒有 `Deployment`，需要手動提交：

```
git add vminsert-deployment.yaml  
git commit -m "轉換 vminsert 為 Deployment"  
git push origin main
```

✓ 步驟 2：強制 ArgoCD 重新同步

讓 ArgoCD 手動檢測 Git 變更：

```
argocd app sync <app-name>
```

如果 ArgoCD UI 仍然沒顯示 `Deployment`，你可以執行：

```
argocd app get <app-name>
```





看看 ArgoCD 是否有發現新的 `Deployment`。

✓ 步驟 3：檢查 ArgoCD `Application` 是否監測 `Deployment`

如果你的 ArgoCD `Application` 設定錯誤，可能它並沒有監測 `Deployment`，可以透過以下方式檢查：

執行：

```
argocd app get <app-name> -o yaml | grep -A5 "source:"
```

如果輸出如下：

```
source:  
  repoURL: https://gitlab.com/your-repo.git  
  targetRevision: main  
  path: k8s/statefulsets # ✗ 這裡錯了，應該要包含 Deployment
```

這代表 ArgoCD 只監測 `statefulsets` 目錄，而 `Deployment` 可能放在 `k8s/deployments` 目錄內，所以 ArgoCD 無法檢測到。

✓ 解決方法 修改 `Application` 的 `path`，確保它監測 `Deployment`：

```
source:  
  repoURL: https://gitlab.com/your-repo.git  
  targetRevision: main  
  path: k8s # ✓ 應該監測 `k8s` 目錄內所有資源
```

然後重新同步：

```
argocd app sync <app-name>
```





✓ 步驟 4：檢查 `Deployment` YAML 是否正確

如果 `Deployment` 在 Git 內已提交，但 ArgoCD 還是沒發現它，可能是 **YAML 配置錯誤**。

📌 檢查 `Deployment` 配置

```
kubectl apply --dry-run=client -f vminsert-deployment.yaml
```

如果有語法錯誤，會顯示：

```
error: error parsing vminsert-deployment.yaml
```

請修正錯誤後，再重新提交 Git。

最終確認

1. 確保 Git 內已刪除 `StatefulSet`，並提交 `Deployment`

```
git add .
git commit -m "移除 StatefulSet，改為 Deployment"
git push origin main
```

2. 手動同步 ArgoCD

```
argocd app sync <app-name>
```

3. 確保 Kubernetes 內的 `Deployment` 正確執行

```
kubectl get deployment -n victoria-metrics
```

4. 確認 ArgoCD UI

- `Sync Status` → 應該變成 `Synced` ✓





- `Health Status` → 應該變成 `Healthy` ✅
-

總結

🚀 當 ArgoCD `OutofSync`，並且

- 顯示 `StatefulSet` 不存在於 Git，但還記得它：手動刪除 `StatefulSet`，然後同步 ArgoCD (`argocd app sync`)。
- 沒有偵測到新的 `Deployment`：
 1. 確保 Git 內有 `Deployment`，並已提交 (`git push`)。
 2. 手動同步 ArgoCD (`argocd app sync`)，讓它更新資源。
 3. 確認 `Application` 的 `path` 是否正確，確保它監測 `Deployment`。

這樣 ArgoCD UI 會變成 `Synced`，`Deployment` 會正常取代 `StatefulSet`，問題解決！🎉



以繁體中文詳細說明:argocd app get <app-name>



ArgoCD 指令 `argocd app get <app-name>` 詳細說明

1. `argocd app get <app-name>` 是什麼？

`argocd app get <app-name>` 是 **ArgoCD CLI** 的一個指令，用來查看 **指定應用** (`Application`) 的當前狀態。

當你執行這個指令時，它會返回該應用的 **同步狀態** (`Sync Status`)、**健康狀態** (`Health Status`)、**部署中的資源** (`Resources`)、**同步歷史** (`History`) 等資訊，幫助你診斷應用是否正





常。

2. `argocd app get <app-name>` 回傳的資訊

當你執行：

```
argocd app get <app-name>
```

它會返回類似這樣的輸出：

```
Name:           victoriametrics-cluster
Project:        default
Server:         https://kubernetes.default.svc
Namespace:      victoria-metrics
URL:            https://argocd.example.com/applications/victoriametrics-
cluster
Repo:           https://gitlab.com/your-repo.git
Target:          main
Path:            k8s
Sync Policy:    Automated (Prune=true, SelfHeal=true)
Sync Status:    OutOfSync from main (49c4e78)
Health Status:  Healthy
```

這段輸出包含以下重要資訊：

欄位	說明
Name	ArgoCD 內的應用名稱 (`Application`), 例如 `victoriametrics-cluster`
Project	該應用所屬的 ArgoCD 項目 (`Project`), 預設為 `default`
Server	Kubernetes API Server 地址
Namespace	該應用部署的 Kubernetes 命名空間





欄位	說明
URL	在 ArgoCD UI 中可以查看此應用的連結
Repo	這個應用的 Git 儲存庫 (`GitLab/GitHub`)
Target	Git 目標分支，例如 `main`
Path	Git 內 YAML 存放的路徑，例如 `k8s/`
Sync Policy	是否啟用了自動同步 (`Automated`)、是否開啟 `Prune`（刪除 Git 內不存在的資源）
Sync Status	同步狀態 (`Synced` 或 `OutOfSync`)
Health Status	應用健康狀態 (`Healthy`, `Degraded`, `Missing`, `Suspended`)

3. `Sync Status` (同步狀態) 解析

當 ArgoCD 檢查你的 **Git Repo** 與 **Kubernetes** 內的實際資源時，它會回報 **同步狀態** (`Sync status`)，主要有以下幾種可能：

Sync Status	說明	解決方案
✓ Synced	Kubernetes 內的配置 與 Git 內完全相符	不需處理，一切正常
⚠ OutOfSync	Kubernetes 內的資源 與 Git 內不同步	執行 `argocd app sync <app-name>` 讓 ArgoCD 套用 Git 內的最新設定
🔴 Unknown	ArgoCD 無法讀取應用的狀態	檢查 ArgoCD 連線是否正常 (`argocd cluster list`)

📌 常見的 `OutOfSync` 原因

1. 你 **手動修改了 Kubernetes** 內的資源，但 **Git** 內 **YAML** 沒有更新。
2. 你 **更新了 Git** 內的 **YAML**，但 **ArgoCD** 還沒同步。



3. 你刪除了某個 Git 內的資源，但 Kubernetes 內仍然存在（例如 `StatefulSet` → `Deployment`）。

4. `Health Status`（健康狀態）解析

ArgoCD 也會回報應用的 健康狀態（`Health Status`），用來判斷應用是否 運行正常。

Health Status	說明	解決方案
Healthy	所有資源運行正常	不需處理
Progressing	部署進行中（新 Pod 啟動、舊版本滾動更新）	稍等幾分鐘後再檢查
Degraded	應用異常，例如 Pod CrashLoopBackOff	使用 `kubectl describe pod` 查錯誤
Missing	Git 內的某些資源不存在於 Kubernetes	執行 `argocd app sync <app-name>` 修復

📌 常見的 `Degraded` 原因

- `Deployment` 或 `StatefulSet` 的 Pod 無法啟動（例如鏡像（`image`）不存在）。
- `Service` 無法找到對應的 Pod，導致 Liveness Probe 失敗。
- `Pod` 進入 CrashLoopBackOff，表示應用程式發生錯誤。

5. `argocd app get <app-name>` 的進階用法

你也可以使用 `--o` 選項，讓 ArgoCD 輸出不同格式的結果：

(1) 以 YAML 格式顯示

如果你想要更詳細的資訊（例如 `spec` 配置），可以加上 `--o yaml`：



```
argocd app get <app-name> -o yaml
```

📌 這會輸出完整的 YAML，包含所有應用的配置

(2) 以 JSON 格式顯示

如果你要把 ArgoCD 資訊 轉換為 JSON，方便程式讀取：

```
argocd app get <app-name> -o json
```

📌 適合與 `jq` 搭配使用，例如

```
argocd app get <app-name> -o json | jq '.status.sync.status'
```

這樣可以 只取出 `Sync Status` 的值。

6. 如何根據 `argocd app get` 的結果修復應用

當你發現 `Sync Status` 是 `OutOfSync`，或者 `Health Status` 是 `Degraded`，你可以採取以下行動來修復：

狀況	解決方案
`OutOfSync`	執行 `argocd app sync <app-name>`，讓 ArgoCD 重新套用 Git 內的設定
`Degraded`	使用 `kubectl describe pod` 查找錯誤，並修正 YAML
`Missing`	如果 `prune` 未啟用，手動刪除 Kubernetes 內的舊資源 (`kubectl delete`)
`Progressing`	稍等幾分鐘，查看 `kubectl get pods` 是否正常啟動





7. 總結

✓ `argocd app get <app-name>` 是用來檢查 ArgoCD 應用的當前狀態。

✓ 它會顯示：

- `Sync Status` (同步狀態)
- `Health Status` (健康狀態)
- **Git 儲存庫 (`Repo`)、目標分支 (`Target`)、監測路徑 (`Path`)。**

✓ 當 `Sync Status` 是 `OutOfSync`，應該執行：

```
argocd app sync <app-name>
```

✓ 當 `Health Status` 是 `Degraded`，應該執行：

```
kubectl describe pod -n <namespace>
```

✓ 如果 ArgoCD 沒有偵測到 `Deployment`，應該檢查：

```
argocd app get <app-name> -o yaml | grep -A5 "source:"
```

✓ 如果 `StatefulSet` 已從 Git 刪除，但 ArgoCD 還記得它，應該手動刪除：

```
kubectl delete statefulset vminsert -n victoria-metrics
```

✓ 可以使用 `-o yaml` 或 `-o json` 來獲取更詳細的資訊：

```
argocd app get <app-name> -o yaml
```

這樣，你就能快速診斷 ArgoCD `OutOfSync` 問題，並確保 `Deployment` 正常運行！🚀





以繁體中文詳細說明:argocd的ui的StatefulSet資訊顯示"
Resource is not found in cluster:
app/v1/StatefulSet:dev-victoriametricsmonitoring-v3"



ArgoCD UI 顯示 `"Resource is not found in cluster: app/v1/StatefulSet:dev-victoriametricsmonitoring-v3"` 詳細解析與解決方法

當你從 `StatefulSet` 轉換為 `Deployment`，但 ArgoCD UI 在 `statefulSet` 區塊顯示 `"Resource is not found in cluster"`，這代表：

1. ArgoCD 仍然認為 `StatefulSet` 應該存在，但 Kubernetes 內已經找不到該資源。
2. 這個 `StatefulSet` 仍然在 ArgoCD 內部的狀態資料庫，但 Kubernetes 內已經不存在了。
3. ArgoCD 尚未完全同步應用，或 `prune`（自動刪除不在 Git 內的資源）未啟用。

這種情況通常發生於：

- 你手動刪除了 Kubernetes 內的 `StatefulSet`，但 ArgoCD 仍然記住它。
- 你已經在 Git 內移除了 `StatefulSet` 配置，改為 `Deployment`，但 ArgoCD 仍然認為 `StatefulSet` 應該存在。

1. 為什麼會發生這個錯誤？

ArgoCD 會監控 Git 內的 Kubernetes 配置（例如 `StatefulSet`、`Deployment`、`Service` 等），並同步到 Kubernetes 叢集。但當你 **刪除或修改** `StatefulSet` 時，ArgoCD 可能會遇到以下情境：





情境	發生原因	ArgoCD UI 顯示狀態
你手動刪除了 Kubernetes 內的 `StatefulSet`	`kubectl delete statefulset` 但 ArgoCD 仍然記錄該資源	`Resource is not found in cluster`
你在 Git 內刪除了 `StatefulSet`，但 ArgoCD 沒有同步	ArgoCD 仍然記住舊的 `StatefulSet`	`outOfSync`
你修改 Git 內的 `StatefulSet` 為 `Deployment`，但 ArgoCD 沒有自動刪除舊的 `StatefulSet`	`prune` 未啟用，ArgoCD 仍然記錄 `StatefulSet`	`Resource is not found in cluster`

📌 核心問題：

- Kubernetes 內 `StatefulSet` 已經刪除，但 ArgoCD 仍然認為它應該存在。
- 如果 `prune` 未啟用，ArgoCD 不會自動刪除 Kubernetes 內不在 Git 內的資源。

2. 如何解決 `Resource is not found in cluster` 問題

✓ 步驟 1：手動同步 ArgoCD

首先，確保 ArgoCD 嘗試重新同步應用，讓它發現 `StatefulSet` 已經被刪除：

```
argocd app sync <app-name>
```

- ◆ 如果 `prune: true` 已啟用，這個指令會讓 ArgoCD 自動刪除 Kubernetes 內不在 Git 內的資源。
- ◆ 如果 `prune: true` 未啟用，則你需要手動刪除 `StatefulSet`（參考步驟 3）。

✓ 步驟 2：確認 `StatefulSet` 是否還存在





執行以下指令，檢查 `StatefulSet` 是否仍然存在：

```
kubectl get statefulset -n <namespace>
```

如果輸出為 空白，代表 `StatefulSet` 已經刪除，但 ArgoCD 仍然記住它：

```
No resources found in <namespace> namespace.
```

✓ 步驟 3：手動刪除 ArgoCD 內部的 `StatefulSet`

如果 ArgoCD 仍然記錄 `StatefulSet`，但實際上 Kubernetes 內已經沒有該資源，你需要手動從 ArgoCD 內部刪除：

```
argocd app delete-resource <app-name> --kind StatefulSet --resource-name dev-victoriametricsmonitoring-v3 --namespace <namespace>
```

這個指令會 強制刪除 ArgoCD 內部狀態中的 `StatefulSet`，讓它不再嘗試同步這個資源。

如果 `argocd app delete-resource` 不可用，你可以試試 `kubectl delete`：

```
kubectl delete statefulset dev-victoriametricsmonitoring-v3 -n <namespace>
```

✓ 步驟 4：啟用 `prune`，讓 ArgoCD 自動刪除 Git 內不存在的資源

如果你希望 以後當 Git 內刪除某個資源時，ArgoCD 會自動刪除 Kubernetes 內的對應資源，請確保 `prune: true` 已啟用。

📌 修改 `Application` 的 `syncPolicy`，確保 ArgoCD 會自動刪除多餘資源





```
syncPolicy:  
  automated:  
    prune: true # 這會讓 ArgoCD 自動刪除 Git 內不存在的 StatefulSet  
    selfHeal: true # 自動修復不同步的資源
```

然後執行：

```
argocd app sync <app-name>
```

✓ 步驟 5：檢查 ArgoCD 的 `Application` 設定

如果 ArgoCD 仍然記錄 `StatefulSet`，可以檢查它的 `Application`：

```
argocd app get <app-name> -o yaml | grep -A5 "source:"
```

看看是否還有 `StatefulSet`，如果有，應該確保 **Git 內 YAML 沒有 `StatefulSet`**，然後重新同步：

```
argocd app sync <app-name>
```

3. 最終確認

✓ 檢查 ArgoCD UI

- 確認 `StatefulSet` 不再出現在 UI 上。
- `Sync Status` 變成 `Synced`。
- `Health Status` 變成 `Healthy`。





✓ 檢查 Kubernetes 內的資源

確保 只有 `Deployment`，沒有 `StatefulSet`：

```
kubectl get statefulset -n <namespace>
kubectl get deployment -n <namespace>
```

應該只看到 `Deployment`：

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
dev-victoriametricsmonitoring-v3	3/3	3	3	10m

✓ 檢查 ArgoCD 是否仍然記住 `StatefulSet`

```
argocd app get <app-name> -o yaml | grep "StatefulSet"
```

如果輸出為空，表示 **ArgoCD** 已經完全刪除 `statefulSet` 的記錄，問題解決！

4. 總結

🚀 當 ArgoCD UI 顯示 `"Resource is not found in cluster: app/v1/StatefulSet"`，表示：

- `StatefulSet` 在 Kubernetes 內已經刪除，但 ArgoCD 仍然記住它。
- 解決方法：
 1. 手動同步 ArgoCD (`argocd app sync <app-name>`)
 2. 檢查 `StatefulSet` 是否仍然存在 (`kubectl get statefulset`)
 3. 如果 Kubernetes 內沒有 `StatefulSet`，手動從 ArgoCD 刪除 (`argocd app delete-resource`)
 4. 啟用 `prune: true`，確保 ArgoCD 會自動刪除 Git 內不存在的資源





5. 檢查 `argocd app get <app-name> -o yaml`，確保 `StatefulSet` 不再被追蹤

这样就能確保 `StatefulSet` 正確移除，`Deployment` 正確運行，ArgoCD UI 不會再顯示錯誤！



以繁體中文詳細說明:我的公司使用Helm，比較這3種方式:
方法 適用場景 優點 缺點

方法 1：直接寫在 args vmstorage 節點固定，簡單環境 易於理解，適合小型集群 變更 vmstorage 需更新 Deployment

方法 2：使用 ConfigMap vmstorage 節點可能變更，需集中管理 易於維護，可即時修改需要多管理 ConfigMap

方法 3：使用環境變數 env 配合 CI/CD，vmstorage 節點不固定 可動態管理變數，適合自動化 變數長時不易閱讀



Helm 部署 `vminsert`：比較 `--storageNode` 設定的 3 種方式

背景

在 VictoriaMetrics Cluster 中，`vminsert` 需要透過 `--storageNode` 參數來指定 vmstorage 節點地址，確保數據能夠正確寫入 `vmstorage`。

在 Helm 部署 `vminsert` 時，有 3 種常見方式來管理 `--storageNode` 參數：

1. 方法 1：直接寫在 `args`
2. 方法 2：使用 `ConfigMap`
3. 方法 3：使用環境變數 (`env`)





1. 方法 1：直接寫在 args

📌 適用場景

- **vmstorage** 節點是固定的，不會變更
- 適用於小型集群，配置簡單，不需要經常變更
- **Helm Chart** 內的 `values.yaml` 直接指定 `args`

✓ 優點

1. **易於理解**：所有 `args` 參數都直接寫在 `values.yaml`，不用額外管理 `ConfigMap`。
2. **適合小型環境**：如果 `vmstorage` 節點是固定的，這種方式最直接。
3. **Helm Chart 結構簡單**：不需要額外建立 `ConfigMap` 或 `Secret`。

✗ 缺點

1. **每次變更 `vmstorage` 節點，都需要重新部署 `vminsert`**
 - 修改 `values.yaml` → 執行 `helm upgrade`，才能讓 `Deployment` 重新載入新設定。
2. **不適合 `vmstorage` 會變更的環境**
 - 如果 `vmstorage` 節點頻繁變更，每次都要 `helm upgrade`，維護成本高。

🚀 Helm Chart 配置

`values.yaml`

```
vminsert:  
  args:  
    - "--storageNode=vmstorage-0.vmstorage:8482"  
    - "--storageNode=vmstorage-1.vmstorage:8482"
```

`deployment.yaml`

```
args:  
  {{- range .Values.vminsert.args -}}
```





```
- {{ . }}
```

```
{{- end }}
```

🎯 適合使用方法 1 的環境

- 小型集群 (`vmstorage` 節點是固定的)
- 變更頻率低 (不常更新 `vmstorage`)
- 簡單易讀 (不想管理 `ConfigMap`)

2. 方法 2：使用 ConfigMap

✖ 適用場景

- `vmstorage` 節點可能變更，需要集中管理
- 適用於中型至大型集群
- 希望不重啟 `vminsert` 就能修改 `storageNode` 設定

✓ 優點

1. 集中管理 `vmstorage` 節點設定：

- `ConfigMap` 讓 `vminsert` 讀取 `--storageNode` 參數時，不需要重新部署 `Deployment`。

2. 變更更靈活：

- 只要更新 `ConfigMap`，`vminsert` 就能重新載入 `--storageNode` 參數，而不需要 `helm upgrade`。

3. 適合 `vmstorage` 變更頻繁的環境。

✗ 缺點

1. 需要額外管理 `ConfigMap`



- `helm upgrade` 時需要確保 `ConfigMap` 正確更新。

2. `vminsert` 需要重新啟動，才能加載新的 `ConfigMap`

- **解決方案**：可以搭配 `emptyDir` 或 `subPath`，讓 `ConfigMap` 熱加載。

🚀 Helm Chart 配置

`values.yaml`

```
vminsert:  
  config:  
    storageNodes: "--storageNode=vmstorage-0.vmstorage:8482 --  
    storageNode=vmstorage-1.vmstorage:8482"
```

`configmap.yaml`

```
apiVersion: v1  
kind: ConfigMap  
metadata:  
  name: vminsert-config  
data:  
  storage-nodes: "{{ .Values.vminsert.config.storageNodes }}"
```

`deployment.yaml`

```
env:  
  - name: STORAGE_NODES  
    valueFrom:  
      configMapKeyRef:  
        name: vminsert-config  
        key: storage-nodes  
args:  
  - "$(STORAGE_NODES)"
```

🎯 適合使用方法 2 的環境

- **vmstorage 節點變更頻繁，需要靈活更新**





- Helm Chart 需要集中管理設定
 - 希望 `vminsert` 讀取 ConfigMap，而不硬編碼 `--storageNode`
-

3. 方法 3：使用環境變數 env

📌 適用場景

- CI/CD (如 GitLab Pipeline) 需要動態更新 `vmstorage` 節點
- 適用於大規模 Kubernetes 叢集
- `vmstorage` 節點經常變更，需要更靈活的管理方式

✓ 優點

1. 可透過 CI/CD 設定 `vmstorage`
 - 例如 GitLab Pipeline 可動態設定 `STORAGE_NODES` 變數。
2. 適合 `vmstorage` 節點會自動變更的情境
 - `Deployment` 會讀取環境變數，而不是寫死 `--storageNode` 參數。
3. 適合 Kubernetes Operator 或 Service Discovery
 - `env` 變數可以來自 DNS SRV 查詢 或 外部 API。

✗ 缺點

1. 變數過長時，不易閱讀
 - `--storageNode` 參數可能會變成很長的環境變數，不易管理。
2. 需要 CI/CD 支援
 - 例如 GitLab 或 ArgoCD 需要提供 `STORAGE_NODES` 變數。

🚀 Helm Chart 配置



`values.yaml`

```
vminsert:  
  env:  
    STORAGE_NODES: "vmstorage-0.vmstorage:8482,vmstorage-1.vmstorage:8482"
```

`deployment.yaml`

```
env:  
  - name: STORAGE_NODES  
    value: "{{ .Values.vminsert.env.STORAGE_NODES }}"  
args:  
  - "--storageNode=$(STORAGE_NODES)"
```

适合使用方法 3 的环境

- 大规模 **Kubernetes** 簇集
- **CI/CD** 需要动态更新 `vmstorage` 节点
- 希望 `vmstorage` 由 **DNS SRV** 或 **API** 来动态发现

4. 三种方法比较

方法	适用场景	优点	缺点
方法 1：直接写在 `args`	`vmstorage` 节点固定，简单环境	易于理解，适合小型集群	变更 `vmstorage` 需重新部署 `vminsert`
方法 2：使用 ConfigMap	`vmstorage` 节点可能变更，需集中管理	易于维护，可即时修改	需要多管理 `ConfigMap`，变更后需重启 Pod
方法 3：使用 `env` 环境变量	CI/CD，`vmstorage` 节点不固定	适合自动化，变更时不影响 Pod	变数过长时不易阅读





5. 結論

- 如果 `vmstorage` 是固定的，選 `方法 1 (args)` → 適合小型集群。
- 如果 `vmstorage` 需要集中管理，選 `方法 2 (ConfigMap)` → 適合中型集群。
- 如果 `vmstorage` 會動態變更，選 `方法 3 (env)` → 適合 CI/CD + 大型集群。

這樣可以確保 Helm Chart 的 `vminsert` 部署靈活、高效，符合不同的業務需求！🚀



我目前將vmagent的srrape job的 yml放在gitlab ci階段，被要求移到cd階段，怎麼處理



如何將 `vmagent` 的 Scrape Job YAML 從 GitLab CI 階段移到 CD 階段

1. 背景

目前 `vmagent` 的 Scrape Job 設定 YAML 被放在 **GitLab CI** (`ci` 階段)，但你被要求將其移到 **CD** (`cd` 階段)。這通常是由於：

1. **Scrape Job** 屬於部署階段，應在 **應用部署** (`cd`) 階段套用，而非 **CI** 測試階段。
2. **分離 CI & CD 的職責**，避免 CI 過程影響實際部署流程。
3. **Scrape Job** 可能需要與 `vmagent` 版本同步更新，所以應該在 `cd` 階段部署。

2. 你的 GitLab CI/CD 設定可能長這樣





目前你的 ` .gitlab-ci.yml ` 可能包含：

```
stages:
  - build
  - test
  - ci # 這裡原本包含 scrape job 部署
  - cd

deploy_vmagent_scrape_job:
  stage: ci # ❌ 這裡應該改成 cd
  script:
    - kubectl apply -f vmagent-scrape-config.yaml
  only:
    - main
```

這表示 **Scrape Job 的 YAML** 在 CI 階段 (`ci`) 被部署，但現在你被要求 **移到 CD 階段** (`cd`)。

3. 如何修改 ` .gitlab-ci.yml `

✓ (1) 修改 `stage`，將 `deploy_vmagent_scrape_job` 從 `ci` 移到 `cd`

```
stages:
  - build
  - test
  - ci # ❌ 不再放在這裡
  - cd # ✓ 應該移到這個階段

deploy_vmagent_scrape_job:
  stage: cd # ✓ 改為 CD 階段
  script:
    - kubectl apply -f vmagent-scrape-config.yaml
```





```
only:  
  - main
```

- ◆ 這樣 `vmagent-scrape-config.yaml` 會在 CD 階段執行，而不是 CI 階段執行。

✓ (2) 確保 Scrape Job YAML 已被正確存入 Git

Scrape Job 通常是一個 **ConfigMap**，所以應該在 `vmagent-scrape-config.yaml` 內正確設定：

```
apiVersion: v1  
kind: ConfigMap  
metadata:  
  name: vmagent-scrape-config  
  namespace: victoria-metrics  
data:  
  scrape-config.yaml: |  
    global:  
      scrape_interval: 15s  
    scrape_configs:  
      - job_name: "node_exporter"  
        static_configs:  
          - targets: ["node-exporter:9100"]
```

- ◆ 確認此檔案已提交到 **GitLab**

```
git add vmagent-scrape-config.yaml  
git commit -m "移動 vmagent scrape job 到 CD 階段"  
git push origin main
```





✓ (3) 確保 `vmagent` 正確加載 Scrape Job

`vmagent` 需要加載 Scrape Job 的 ConfigMap，確保 `Deployment` 內有這段：

`vmagent-deployment.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vmagent
  namespace: victoria-metrics
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: vmagent
    spec:
      containers:
        - name: vmagent
          image: victoriametrics/vmagent:v1.93.2
          args:
            - "-promscrape.config=/etc/vmagent/scrape-config.yaml"
      volumeMounts:
        - name: scrape-config
          mountPath: /etc/vmagent
          subPath: scrape-config.yaml
      volumes:
        - name: scrape-config
          configMap:
            name: vmagent-scrape-config
```

◆ 這確保 `vmagent` 會讀取 `scrape-config.yaml`，並從 ConfigMap 獲取最新 Scrape Job 設定。

✓ (4) 測試 GitLab CI/CD Pipeline





現在，你可以推送變更，讓 GitLab CD 階段來部署 Scrape Job：

```
git push origin main
```

◆ 檢查 **GitLab CI/CD Pipeline** 進入 GitLab → **CI/CD** → **Pipelines**，應該會看到 **deploy_vmagent_scrape_job** 在 `cd` 階段執行。

◆ 確認 **ConfigMap** 是否正確部署

```
kubectl get configmap -n victoria-metrics
```

應該看到：

NAME	DATA	AGE
vmagent-scrape-config	1	5m

◆ 確認 `vmagent` 是否正確載入 **Scrape Job**

```
kubectl logs -l app=vmagent -n victoria-metrics
```

如果 `vmagent` 啟動成功，應該會顯示：

```
loading configuration from /etc/vmagent/scrape-config.yaml
```

這表示 **Scrape Job** 設定已成功載入！

4. 總結





修改項目	原本 (CI 階段)	修改後 (CD 階段)
GitLab CI/CD `stage`	`stage: ci`	`stage: cd`
執行時機	CI 階段	CD 階段
ConfigMap	可能未使用	使用 ConfigMap
`vmagent` 配置	可能未正確加載 Scrape Job	`volumeMounts` 讀取 Scrape Job

🚀 這樣可以確保 `vmagent` Scrape Job 只在 CD 階段部署，並確保設定變更時自動同步！ 🎯



以繁體中文詳細說明：這個Config map可以給不同的vmagent共同使用嗎



`ConfigMap` 是否可以給不同的 `vmagent` 共同使用？

1. `ConfigMap` 是否能共享？

✅ 可以共享！

在 Kubernetes 中，`ConfigMap` 是命名空間 (`namespace`) 內的共享資源，所以：

- 只要 `ConfigMap` 在相同的 `namespace`，多個 `vmagent` 可以共用同一個 `ConfigMap`。
- 不同 `namespace` 的 `vmagent`，需要在各自的 `namespace` 內建立相同的 `ConfigMap`，或是用 `kustomize` / `Helm` 來動態配置。





2. 如何讓多個 `vmagent` 共同使用相同的 `ConfigMap`？

✓ 方式 1：在相同 `namespace` 內共用 `ConfigMap`

如果 所有 `vmagent` 都在相同的 `namespace`，你可以 直接讓所有 `vmagent` 掛載相同的 `ConfigMap`。

📌 `ConfigMap` (`vmagent-scrape-config.yaml`)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: vmagent-scrape-config
  namespace: victoria-metrics # 確保所有 vmagent 都在這個 namespace
data:
  scrape-config.yaml: |
    global:
      scrape_interval: 15s
    scrape_configs:
      - job_name: "node_exporter"
        static_configs:
          - targets: ["node-exporter:9100"]
```

📌 `Deployment` (`vmagent-deployment.yaml`)

✓ 讓所有 `vmagent` 掛載相同的 `ConfigMap`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vmagent-1
  namespace: victoria-metrics
spec:
  replicas: 2
  selector:
    matchLabels:
      app: vmagent
  template:
    metadata:
```





```
labels:
  app: vmagent
spec:
  containers:
    - name: vmagent
      image: victoriametrics/vmagent:v1.93.2
      args:
        - "-promscrape.config=/etc/vmagent/scrape-config.yaml"
  volumeMounts:
    - name: scrape-config
      mountPath: /etc/vmagent
      subPath: scrape-config.yaml
  volumes:
    - name: scrape-config
      configMap:
        name: vmagent-scrape-config # ✅ 讓所有 vmagent 掛載相同的 ConfigMap
```

📌 這樣，所有 `vmagent`（例如 `vmagent-1`，`vmagent-2`）都可以讀取 `vmagent-scrape-config`，使用相同的 **Scrape Job 設定**！

✅ 方式 2：不同 `namespace`，用 `Helm` 或 `Kustomize` 部署相同 `ConfigMap`

如果 **不同的 `vmagent` 在不同的 `namespace`**，那麼：

1. **ConfigMap 不能跨 `namespace` 共用**（每個 `namespace` 內的 `ConfigMap` 是獨立的）。
2. **解決方案：使用 `Helm` 或 `Kustomize`，在多個 `namespace` 內部署相同的 `ConfigMap`**。

📌 `Helm values.yaml`

```
vmagent:
  namespaces:
    - victoria-metrics
    - monitoring
  scrapeConfig:
```





```
storageNodes: "--storageNode=vmstorage-0.vmstorage:8482 --  
storageNode=vmstorage-1.vmstorage:8482"
```

📌 `Helm template` (`configmap.yaml`)

```
{{- range .Values.vmagent.namespaces }}  
apiVersion: v1  
kind: ConfigMap  
metadata:  
  name: vmagent-scrape-config  
  namespace: {{ . }}  
data:  
  scrape-config.yaml: |  
    global:  
      scrape_interval: 15s  
      scrape_configs:  
        - job_name: "node_exporter"  
          static_configs:  
            - targets: ["node-exporter:9100"]  
    ---  
{{- end }}
```

✓ 這樣 `Helm` 會在 `victoria-metrics` 和 `monitoring` 兩個 `namespace` 內，部署相同的 `ConfigMap`，確保不同 `vmagent` 可以共用相同的 Scrape Job！

3. 什麼情況下 `ConfigMap` 不能共用？

✗ 不同 `namespace` 無法直接共用 `ConfigMap`：

- **ConfigMap 是 `namespace` 限定的**，`vmagent` 不能直接掛載不同 `namespace` 內的 `ConfigMap`。
- **解決方案**：使用 `Helm` 或 `Kustomize` 在不同 `namespace` 內創建相同的 `ConfigMap`。

✗ 如果 `vmagent` 需要不同的 Scrape Job 設定：





- **解決方案**：可以在 `Helm` 或 `Kustomize` 內，針對不同 `vmagent` 動態配置不同的 Scrape Job。

4. 結論

方案	適用情境	優點	缺點
相同 `namespace` 共用 `ConfigMap`	所有 `vmagent` 都在同一個 `namespace`	簡單易維護，變更即時生效	不能跨 `namespace` 使用
使用 `Helm` 部署到多個 `namespace`	`vmagent` 分佈在不同 `namespace`	自動化部署，適合 CI/CD	需要 `Helm` 或 `Kustomize`
不同 `vmagent` 使用不同的 Scrape Job 設定	每個 `vmagent` 有不同的監控需求	可獨立管理不同 `vmagent`	需要動態管理 `ConfigMap`

🚀 建議

- 如果 `vmagent` 在相同 `namespace`，直接共用 `ConfigMap` ✓
- 如果 `vmagent` 在不同 `namespace`，用 `Helm` 部署多個 `ConfigMap` ✓
- 如果不同 `vmagent` 需要不同 Scrape Job，則建立不同的 `ConfigMap` ✓

這樣可以確保 `vmagent` 在 Kubernetes 內高效運行，並靈活管理 Scrape Job ! 🎉

