

vmalert Alertmanager 設定



以繁體中文詳細說明:我用dockercomposey在虛擬機建好 vminsert、vmselect、vmstorage、vmagent、vmalert 了，請問怎麼用vmalert + alertmanager設定alert

我想要理解vmalert + alertmanager用法，為了做 vminsert、vmselect、vmstorage、vmagent、vmaler cpu使用率超過10%發出mail通知我的實驗，我對vmalert與 alertmanager用法不熟，你能教我嗎，或是你能給更簡單的範例：



當然可以！以下會用**繁體中文**詳細說明 **vmalert + Alertmanager** 如何在 VictoriaMetrics 架構下設定 **CPU 使用率超過 10% 發送 Email 警報**。我會先解釋原理、組成，再給你**最簡單可執行的範例**，最後補充各步驟的注意事項。

一、整體架構與運作流程

1. **vmalert**：類似 Prometheus Alertmanager 的 alert rule evaluator。負責根據你定義的規則查詢 VictoriaMetrics（可經過 vmselect），若達到條件則發出 Alert。
2. **Alertmanager**：接收 vmalert 發送的 Alert，負責分發通知（如 Email、Line、Slack、Webhook...）。

流程說明

- vmalert 定時執行 PromQL 查詢（通常 query vmselect）。
- 若符合警戒條件，vmalert 觸發告警，發送 alert 到 Alertmanager。



- Alertmanager 依據你設定的 receiver，將警報轉發（如 Email）。

二、vmaalert + Alertmanager 設定範例

1. 定義 vmaalert 規則 (rule file)

建立一個 `rules.yml`，內容如下，這邊以 CPU 使用率（假設 metric 名稱為 `process_cpu_seconds_total`）為例：

```
groups:
  - name: cpu_usage_rules
    rules:
      - alert: HighCPUUsage
        expr:
          (rate(process_cpu_seconds_total{job=~"vminsert|vmselect|vmstorage|vmagent|vmalert"}[5m])) * 100 > 10
        for: 1m
        labels:
          severity: critical
        annotations:
          summary: "{{ $labels.job }}" CPU 使用率超過 10%"
          description: "{{ $labels.job }}" CPU 使用率過高，目前為 {{ $value }}%"
```

“注意：”

- “`process_cpu_seconds_total` 是最常見的 exporter cpu metric。實際請確認你容器/虛擬機暴露的 metric 名稱。”
- “`{job=~"..."}` 代表多個 job。”
- “`rate(...[5m]) * 100` 粗略近似百分比（前提是有以 process exporter 暴露這個 metric）。”

2. vmalert docker-compose 設定

```
vmalert:
  image: victoriametrics/vmalert
  container_name: vmalert
  command:
    - '-datasource.url=http://vmselect:8481/select/0/prometheus'
    - '-rule=/rules/rules.yml'
    - '-notifier.url=http://alertmanager:9093'
    - '-external.url=http://你的vmalert對外網址:8880'
  volumes:
    - ./rules.yml:/rules/rules.yml
  ports:
    - "8880:8880"
  depends_on:
    - vmselect
    - alertmanager
```

3. Alertmanager docker-compose 設定 & 配置

Alertmanager 部分：

```
alertmanager:
  image: prom/alertmanager
  container_name: alertmanager
  ports:
    - "9093:9093"
  volumes:
    - ./alertmanager.yml:/etc/alertmanager/alertmanager.yml
```

Email 通知設定 (alertmanager.yml 範例)：

```
global:
  smtp_smarthost: 'smtp.gmail.com:587'
```

```
smtp_from: 'your_gmail@gmail.com'
smtp_auth_username: 'your_gmail@gmail.com'
smtp_auth_password: 'your_app_password' # 建議用Gmail應用程式密碼
smtp_require_tls: true

route:
  receiver: 'email-alert'
receivers:
  - name: 'email-alert'
    email_configs:
      - to: '你的收信信箱@gmail.com'
        send_resolved: true
```

“注意 **Gmail** 密碼需用「應用程式專用密碼」取得，不能直接用登入密碼！”

4. 重啟 Docker Compose

假設你的 `docker-compose.yml` 都寫好，上述 rules.yml、alertmanager.yml 也正確：

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

5. 測試/驗證

1. **vmaalert** 網頁介面 (<http://你的IP:8880/>)：可以看到 Alert 狀態。
2. **Alertmanager** 網頁介面 (<http://你的IP:9093/>)：可以看到已發出的 alert。
3. **Email** 信箱：收信，確認可收到警報。

三、常見問題與補充

- **metrics 名稱對不上？**

用 ``curl http://你的vmagent:8429/metrics`` 或 ``curl http://你的vmselect:8481/select/0/prometheus/api/v1/label/__name__/values`` 查詢可用的 metrics。

- **警報沒有觸發？**

1. 檢查 PromQL 是否有資料（Grafana 或 API 查詢）。
2. 檢查 rule file 語法、job 名稱、label 是否正確。
3. 查看 vmaalert logs。

- **Alertmanager Email 發不出去？**

1. 檢查 smtp 設定。
2. 查看 alertmanager logs。
3. Gmail 一定要用應用程式密碼。

四、延伸閱讀/參考文件

- [VictoriaMetrics vmaalert 官方文件](#)
- [Alertmanager 官方文件](#)
- [VictoriaMetrics vmaalert alerting_rule](#)
- [Prometheus process exporter metrics](#)

五、超簡易總結流程

1. 寫好 **rules.yml** (用 rate + job label 選擇你要監控的元件)
2. 設定 **vmaalert** 指向 **vmselect & alertmanager**
3. 設定 **alertmanager** 發送 **email**
4. 重啟服務並測試

如果你有任何一個步驟遇到問題 (metrics 名稱、docker-compose 配置、PromQL 怎麼寫)，都可以直接貼出你的設定或錯誤訊息，我再幫你針對性 debug !

要更簡化範例或有實際 log 想排查也隨時說 !



以繁體中文詳細說明:1. 定義 vmaalert 規則 (rule file)、3. Alertmanager docker-compose 設定 & 配置



當然可以！我來用繁體中文詳細說明：

1. 定義 vmaalert 規則 (rule file)

vmaalert 的警報規則格式與 Prometheus 的 Alerting Rule 幾乎一樣，通常以 YAML 格式儲存 (例如 ``rules.yml``)。

範例說明

假設你要監控 vminsert、vmselect、vmstorage、vmagent、vmaalert 這幾個元件的 **CPU 使用率**，若大於 10% 持續 1 分鐘就觸發警報。

rules.yml 範例：

```
groups:
  - name: VictoriaMetrics CPU 使用率監控
    rules:
      - alert: HighCPUUsage
        expr:
(rate(process_cpu_seconds_total{job=~"vminsert|vmselect|vmstorage|vmagent|vmalert"}[5m])) * 100 > 10
        for: 1m
        labels:
          severity: critical
        annotations:
          summary: "{{ $labels.job }}" CPU 使用率過高"
          description: "{{ $labels.job }}" CPU 使用率大於 10%，目前為 {{ $value
}}%"
```

重點說明：

- `groups` → 一組規則（可多組）
- `name` → 群組名稱（自訂）
- `rules` → 多個警報規則
- `alert` → 此規則的警報名稱（自訂）
- `expr` → **PromQL 查詢條件**，計算 cpu 使用率，job 可用 regex 指定多個
- `for` → **持續時間**，達標多久才觸發
- `labels` → 自訂標籤（可用於分類、路由）
- `annotations` → 告警訊息模板，`\$labels.job` 代表 job 名稱、`\$value` 代表當下值

備註：

- `process_cpu_seconds_total` 是常見的 CPU 指標（取決於你的 exporter，有時會叫 `container_cpu_usage_seconds_total` 等，需確認你的 metrics 名稱）

- 如果要測試，可先把 10 改小一點，比如 0.1，方便測試

3. Alertmanager docker-compose 設定 & 配置

Alertmanager 負責接收 vmalert 發來的告警，並根據你設定的路由與 receiver 發送通知（如 email）。

(1) docker-compose 設定範例

假設你的 `alertmanager.yml` 配在本機（同一目錄），docker-compose.yml 可以這樣寫：

```
version: '3.7'
services:
  alertmanager:
    image: prom/alertmanager
    container_name: alertmanager
    ports:
      - "9093:9093"
    volumes:
      - ./alertmanager.yml:/etc/alertmanager/alertmanager.yml
```

- `image`：官方 Alertmanager
- `ports`：9093 是 web ui 預設 port
- `volumes`：把你本地的設定檔掛進去容器

(2) Alertmanager 設定檔（alertmanager.yml）範例

這裡示範發 email，以 Gmail 為例（你可改其他 SMTP）：


```
global:
  smtp_smarthost: 'smtp.gmail.com:587'
  smtp_from: 'your_gmail@gmail.com'
  smtp_auth_username: 'your_gmail@gmail.com'
  smtp_auth_password: 'your_app_password' # 請用Gmail應用程式密碼
  smtp_require_tls: true

route:
  receiver: 'email-notice'

receivers:
  - name: 'email-notice'
    email_configs:
      - to: '你的收信信箱@gmail.com'
        send_resolved: true
```

參數解釋：

- ``smtp_smarthost``：SMTP伺服器（Gmail 用這個）
- ``smtp_from``：寄件人信箱
- ``smtp_auth_username`/`smtp_auth_password``：Gmail帳號/密碼（用「應用程式密碼」）
- ``route.receiver``：預設通知目標（這裡是 email-notice）
- ``receivers``：一組一組通知設定
- ``email_configs.to``：收件人
- ``send_resolved``：警報解除時也寄信

“ Gmail 密碼請不要用一般密碼，必須先開啟 2 階段驗證，再去 Google 帳號安全性申請「應用程式密碼」。”

(3) 啟動/重啟服務

Alertmanager 設定好之後，用下面指令啟動或重啟：

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

或

```
docker restart alertmanager
```

補充

- 你可以用瀏覽器開啟 `http://你的IP:9093/` 進入 Alertmanager UI，看到收到的 alert
- vmalert 要設定 `-notifier.url=http://alertmanager:9093` 來送警報
- 如果 Email 沒收到，可以看一下 Alertmanager 的 log，或測試 SMTP 設定

如還有哪一部分想要進一步範例或 debug 步驟，請直接提出！



以繁體中文詳細說明：

Vmalert之後要用gitlab pipeline佈到k8s環境，應該使用 deployment還是statefulset?



這個問題非常實用，下面我用**繁體中文詳細說明**，並針對 **vmalert 在 GitLab Pipeline 部署到 K8s** 的情境，深入說明你該選用 **Deployment 還是 StatefulSet**。

一、Deployment 與 StatefulSet 差異快速比較

特性	Deployment	StatefulSet
Pod 名稱	隨機產生	有序且固定（如 vmalert-0, vmalert-1）
資料持久化	不適合，需要搭配 PVC 自行管理	適合有狀態、需要唯一存儲
伸縮性	快速，可自動增減無狀態 Pod	有序調整，適合有狀態應用
適合場景	無狀態服務（API、Web、Worker等）	有狀態服務（DB、Kafka、Zookeeper等）

二、vmalert 的特性與需求分析

vmalert 的本質

- **無狀態**：vmalert 本身並不需要本地唯一磁碟、也不需記憶過去狀態（警報狀態會自動重新查詢 data source），而且它通常是靠參數指向 vmselect / data source 以及 alertmanager。
- **高可用（可選）**：有些進階場景會用多個 vmalert 分開跑不同 rule，或用 external tool 做 HA。

所以：

- 你不需要每一個 vmalert 副本有獨立身分（如 DB 叢集需要獨立序號），
 - 也不需要固定的網路名稱、穩定身份或對外掛載唯一的 volume。
-

三、在 GitLab Pipeline 佈署 vmaalert 到 K8s，建議用 Deployment

理由如下：

1. 無狀態適合用 Deployment

- vmaalert 佈署後，Pod 隨時可 scale/out/in，沒有資料綁定特定 Pod。

2. 彈性升級與滾動更新

- Deployment 支援 rolling update、回滾等標準 CI/CD 功能，最適合 pipeline 流程。

3. 不需每個 Pod 都有固定名稱或順序

- StatefulSet 僅適合需要唯一身份、穩定 volume 的狀況（如 DB）。

除非：

- 你有非常特殊需求，每個 vmaalert 都有不同 rule set 或 volume 綁定，否則不建議用 StatefulSet。

四、vmaalert Deployment 範例 YAML

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vmaalert
spec:
  replicas: 1
  selector:
    matchLabels:
      app: vmaalert
  template:
```

```
metadata:
  labels:
    app: vmalert
spec:
  containers:
    - name: vmalert
      image: victoriametrics/vmalert:v1.95.2
      args:
        - '-datasource.url=http://vmselect:8481/select/0/prometheus'
        - '-rule=/rules/rules.yml'
        - '-notifier.url=http://alertmanager:9093'
      volumeMounts:
        - name: rules-volume
          mountPath: /rules
  volumes:
    - name: rules-volume
      configMap:
        name: vmalert-rules # 假設你把 rules.yml 放 configmap
```

- 如需多副本，`replicas: 2`、`3` 即可，無資料衝突疑慮。
- 建議 rules.yml 放 configMap，每次 pipeline 自動同步。

五、結論與部署建議

- **vmalert 佈署到 K8s**：「用 **Deployment** 即可」
- **不用 StatefulSet**，除非有唯一身份、資料、Volume 等特殊需求
- 配合 CI/CD，Deployment 最易整合、維運

如有 GitLab Pipeline YAML 或 vmalert rules 的自動化問題，也可以再詢問，我可以提供更完整的 CI/CD 實作建議！



vmaalert-ha.pdf
PDF

參考vmaalert-ha.pdf，以繁體中文詳細說明:我需要建立vmaalert ha到k8s，我對vmaalert ha用法不熟，你能教我嗎，或是你能給更簡單的範例



根據你提供的 **vmaalert-ha.pdf** 以及 VictoriaMetrics 官方說明，以下以**繁體中文**詳細說明如何在 Kubernetes 建立 **vmaalert HA（高可用）**，並輔以易懂範例。

一、什麼是 vmaalert HA？

vmaalert 高可用就是同時啟動多個配置完全相同的 **vmaalert 實例**，讓這些實例分擔 alert/rule 計算工作，即使其中一個 Pod 掛掉，alert 還是能正常執行與通知，**避免單點故障**。

二、vmaalert HA 架構重點

1. 多個 **vmaalert 實例**，配置完全一樣（rules、datasource、alertmanager...）
2. 所有 **vmaalert** 都對同一組 VictoriaMetrics/vmselect 查詢，寫入同一組 **vminsert**
3. **-notifier.url** 可指定多個 **Alertmanager**
4. **HA** 必要時要設定 **-remoteWrite.url** 與 **-remoteRead.url**，保證 **Alert** 狀態可持久化與恢復



5. 為避免資料重複，需設置 **Deduplication**（去重）相關參數

三、K8s 部署建議

vmalert HA 要使用 **Deployment**，非 **StatefulSet**（因為無狀態）。直接將 **Deployment** 的 **replicas** 設為 2 或 3（可自行決定）。

四、vmalert HA 簡易 K8s Deployment 範例

以下為簡化範例，建議以此為基礎改成你自己的路徑與設定：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vmalert-ha
spec:
  replicas: 2 # 建議至少兩個 Pod 做高可用
  selector:
    matchLabels:
      app: vmalert
  template:
    metadata:
      labels:
        app: vmalert
    spec:
      containers:
        - name: vmalert
          image: victoriametrics/vmalert:v1.95.2
          args:
            - '-rule=/rules/rules.yml'
            - '-datasource.url=http://vmselect:8481/select/0/prometheus'
            - '-notifier.url=http://alertmanager:9093'
```

```

- '-remoteWrite.url=http://vminsert:8480/insert/0/prometheus'
- '-remoteRead.url=http://vmselect:8481/select/0/prometheus'
- '-external.label=replica=$(POD_NAME)' # 可用於分辨不同 replica
- '-dedup.minScrapeInterval=30s' # 依照 rules 設定調整
env:
  - name: POD_NAME
    valueFrom:
      fieldRef:
        fieldPath: metadata.name
  volumeMounts:
    - name: rules
      mountPath: /rules
  volumes:
    - name: rules
      configMap:
        name: vmalert-rules-config

```

- **replicas**：設成 2 或 3 就會自動高可用
- **dedup.minScrapeInterval**：與 rules 的 group interval 取最大公因數，避免重複寫入
- **external.label**：方便排查 HA 各 Pod 狀態

五、Deduplication（去重）機制

若多個 vmalert HA 實例同時運作，**VictoriaMetrics** 會自動根據時間戳去重複（需正確設定 dedup.minScrapeInterval），而 **Alertmanager** 也會自動 deduplicate label 完全相同的 alerts。

六、Alertmanager HA 設定

建議 Alertmanager 也要高可用。你可在 vmalert 指定多個 notifier，例如：


```
- '-notifier.url=http://alertmanager1:9093'
- '-notifier.url=http://alertmanager2:9093'
```

七、完整高可用流程

1. 多個 vmalert Pod 讀同一份 rule，查同一組 VM/vmselect
2. 評估警戒時，每個 Pod 都會計算並發 alert，但只會有一份警報被通知（dedup label 相同則 Alertmanager 只送一次）
3. 即使其中一台 Pod 掛掉，另一台還在，Alert 不會中斷

八、簡單 rules.yml 範例

```
groups:
- name: example_group
  interval: 30s
  rules:
  - alert: VMHighCPU
    expr: process_cpu_seconds_total > 0.1
    for: 1m
    labels:
      severity: critical
    annotations:
      summary: "{{ $labels.instance }}" CPU 過高"
```

“記得 group interval 要與 dedup.minScrapeInterval 配合”

九、補充：可用 **vmaalert HA** 健康檢查

- 查看 `/metrics` endpoint`
 - 查看 vmaalert web UI (預設 8880 port)
 - 查看 Alertmanager UI (9093 port)
-

十、進階參數與參考

- ``-evaluationInterval``：可設定規則計算頻率 (group 可個別 interval 覆蓋)
 - ``-external.label``：標記不同實例
 - ``-rule=/rules/*.yaml``：支援通配符
 - 參考官方[HA 架構圖](#)
-

十一、簡化流程整理

1. **Deployment** 部署 **vmaalert**, 設多個 **replicas**
 2. 配置 **dedup.minScrapeInterval**, 確保不重複寫入
 3. **rules** 設 **interval**, 注意需和 **dedup** 搭配
 4. 可多個 **-notifier.url** 指到多組 **Alertmanager**
 5. **rules.yaml** 放 **configMap** 熱更新更方便
-

如需 **YAML/CI/CD** 自動化範例或遇到任何錯誤訊息，歡迎貼出來，幫你 **debug** 或補上進階解釋！

參考來源

- [VictoriaMetrics vmalert 官方文件](#)

•

資料來源



參考vmalert-ha.pdf，以繁體中文詳細說明:什麼是 vmalert HA、vmalert HA 架構重點、K8s 部署建議、四、vmalert HA 簡易 K8s Deployment 範例、五、Deduplication（去重）機制



當然沒問題！以下是根據 **vmalert-ha.pdf** 與官方說明，以**繁體中文詳細說明**你指定的五個主題。

一、什麼是 vmalert HA？

vmalert HA（高可用）

就是同時運行多個設定完全相同的 vmalert 實例（Pod），讓這些實例**彼此備援**、分攤警報規則運算，即使某個實例發生故障，警報評估與通知依然不中斷，確保整體 alerting 服務不會因單一點故障（SPOF）而中斷。

- **關鍵重點：**每個 vmalert 實例都會根據同一套規則對同一數據源進行查詢、發送告警到同一組 Alertmanager。



二、vmalet HA 架構重點

1. **多實例執行**：至少兩個以上 vmalet 實例，規則 (rule files)、datasource、notifier 參數必須完全相同。
 2. **讀同一個數據源**：多個 vmalet 指向同一個 vmselect (VictoriaMetrics cluster 查詢服務)。
 3. **寫同一個目標**：警報狀態與 recording rule 結果寫入同一個 vminsert。
 4. **多個 Alertmanager**：vmalet 支援設定多組 Alertmanager 位址 (多個 -notifier.url)。
 5. **去重**：VictoriaMetrics 和 Alertmanager 皆自帶 dedup (去重) 機制，確保警報不重複通知。
 6. **持久化狀態**：建議設定 -remoteWrite.url 和 -remoteRead.url，確保警報狀態持久化、重啟可恢復狀態。
 7. **部署用 Deployment**：在 K8s 以 Deployment 搭配多副本 (replicas) 方式運行。
-

三、K8s 部署建議

- **建議用 Deployment** (不是 StatefulSet, 因為 vmalet 是無狀態服務)。
 - replicas 設定為 2 或 3, 即可達到 HA 目標。
 - rule files 建議以 ConfigMap 管理, 方便多個 Pod 共用與熱更新。
 - -dedup.minScrapeInterval 參數需與 rules 設定的 interval 配合, 防止多個實例重複寫入。
-

四、vmalert HA 簡易 K8s Deployment 範例

以下為簡易 YAML 範例（請根據自身需求修改）：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vmalert-ha
spec:
  replicas: 2 # 兩個 Pod, 達到 HA
  selector:
    matchLabels:
      app: vmalert
  template:
    metadata:
      labels:
        app: vmalert
    spec:
      containers:
        - name: vmalert
          image: victoriametrics/vmalert:v1.95.2
          args:
            - '-rule=/rules/rules.yml'
            - '-datasource.url=http://vmselect:8481/select/0/prometheus'
            - '-notifier.url=http://alertmanager:9093'
            - '-remoteWrite.url=http://vminsert:8480/insert/0/prometheus'
            - '-remoteRead.url=http://vmselect:8481/select/0/prometheus'
            - '-dedup.minScrapeInterval=30s' # 根據 rule group interval 設定
          volumeMounts:
            - name: rules
              mountPath: /rules
      volumes:
        - name: rules
          configMap:
            name: vmalert-rules-config # 預設 rules.yml 存於 ConfigMap
```

- **replicas**：設成 2（可依需求調整）

- **-dedup.minScrapeInterval**：設為你所有 rule group interval 的最大公因數（如 30s/60s）
 - **rules**：以 ConfigMap 方式共用，方便多 Pod 共用且熱更新
-

五、Deduplication（去重）機制

為什麼要去重？

多個 vmalet HA 實例會同時產生一樣的警報、寫入一樣的資料。
若不做去重，會造成：

- 時序資料重複寫入 VictoriaMetrics
- Alertmanager 收到重複警報，產生多封通知

怎麼達到去重？

1. VictoriaMetrics 端（存資料時）

- 利用 **deduplication** 機制，自動根據相同的 label set、timestamp 只保留一份資料
- 參數 `-dedup.minScrapeInterval` 控制去重的最小時間間隔，建議設成與規則 group 的 interval 相同或最大公因數

2. Alertmanager 端（發警報時）

- 若多個 vmalet 實例發送完全一樣的 alert（label 相同），Alertmanager 只會發出一份通知
- 保證無論有幾個 vmalet 實例同時觸發，都不會重複通知

設定重點

- 每個 vmalet 都要設 `-dedup.minScrapeInterval`

- rules group interval 請和 dedup.minScrapeInterval 配合
- Alertmanager 自動 dedup，無需額外設定

總結

- vmalert HA：多個設定一樣的 vmalert 實例同時運作，確保高可用
- 架構重點：多實例、同 rules、同數據源、同 Alertmanager，Dedup 機制避免重複
- K8s 建議用 Deployment (replicas ≥ 2)
- Deployment YAML 範例可參考上面內容
- dedup 需正確設定，確保資料與警報不會重複

如需更進階的 dedup 原理或遇到實際錯誤，歡迎隨時再提問！



參考vmalert-ha.pdf，以繁體中文詳細說明:多個 Alertmanager：vmalert 支援設定多組 Alertmanager 位址 (多個 -notifier.url)。

去重：VictoriaMetrics 和 Alertmanager 皆自帶 dedup (去重) 機制，確保警報不重複通知。

持久化狀態：建議設定 -remoteWrite.url 和 -remoteRead.url，確保警報狀態持久化、重啟可恢復狀態。

部署用 Deployment：在 K8s 以 Deployment 搭配多副本 (replicas) 方式運行。



好的，以下根據 **vmaalert-ha.pdf** 以繁體中文詳細說明你的指定重點：

1. 多個 Alertmanager：vmaalert 支援設定多組 Alertmanager 位址（多個 -notifier.url）

說明：

- vmaalert 可以同時設定多個 Alertmanager 通知端點，只要在啟動參數中重複加入 `-notifier.url``。
- 這麼做的目的是**提升警報通知的高可用性**，即使某個 Alertmanager 宕機，其他還能正常接收告警。
- 每一個 fired 的警報都會被**同時送到所有設定的 Alertmanager**，由 Alertmanager HA cluster 做進一步的 dedup 處理。

範例：

```
./vmaalert \  
-rule=/rules/rules.yml \  
-datasource.url=http://vmselect:8481/select/0/prometheus \  
-notifier.url=http://alertmanager1:9093 \  
-notifier.url=http://alertmanager2:9093 \  
...
```

- 如上，只需多加幾個 `-notifier.url`` 即可。
- K8s 內可透過 Service 或 headless Service 實現多實例連線。

2. 去重（Deduplication）：VictoriaMetrics 和 Alertmanager 皆自帶 dedup 機制

VictoriaMetrics :

- 多個 vmaalert HA 實例會重複寫入同樣的警報與 recording rule 結果到 VictoriaMetrics。
- VictoriaMetrics 會依據**相同 label set 和 timestamp**自動去重，只保留一份資料，這就是所謂的 deduplication。
- 須設好 `-dedup.minScrapeInterval`（這個間隔需與警報規則 group 的 interval 相同或其最大公因數），這樣 VictoriaMetrics 才能正確判斷哪些數據要去重。

Alertmanager :

- 多個 vmaalert 觸發的警報若 labels 相同，Alertmanager 會自動 deduplicate，只寄一份警報給使用者，不會造成信件/通知暴增。

注意事項：

- 為確保 dedup 準確生效，**多個 vmaalert 必須使用完全相同的 rules 與 labels**。

3. 持久化狀態：-remoteWrite.url 和 -remoteRead.url

說明：

- vmaalert 的警報（alerting rule）和錄製規則（recording rule）狀態預設**僅存在記憶體**。
- 如未設定持久化，當 vmaalert 重啟時所有警報狀態都會重置，過去有觸發的警報也會遺失。
- 建議**設定 `-remoteWrite.url` 及 `-remoteRead.url`**，將警報狀態與 rule 結果寫入到 VictoriaMetrics（或 vminsert），重啟時自動還原狀態。

範例：

```
./vmaalert \  
... \  

```

```
-remoteWrite.url=http://vminsert:8480/insert/0/prometheus \
-remoteRead.url=http://vmselect:8481/select/0/prometheus \
```

- remoteWrite 會將警報與錄製狀態寫入時序資料庫
- remoteRead 讓 vmalert 重啟時可讀回上次狀態，不會丟失已 firing/pending 的告警

4. 部署用 Deployment : K8s 以 Deployment 搭配多副本 (replicas) 方式運行

說明：

- vmalert 為無狀態服務 (stateless)，建議用 K8s Deployment 來部署。
- 只要把 Deployment 的 `replicas` 設成 2 (或更多)，即可讓多個 vmalert HA 實例同時運作，達到高可用。
- rules.yml、相關設定建議用 ConfigMap 管理，讓所有 Pod 共用相同配置。

簡單範例：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vmalert-ha
spec:
  replicas: 2
  selector:
    matchLabels:
      app: vmalert
  template:
    metadata:
      labels:
        app: vmalert
    spec:
```

```
containers:
  - name: vmalert
    image: victoriametrics/vmalert:v1.95.2
    args:
      - '-rule=/rules/rules.yml'
      - '-datasource.url=http://vmselect:8481/select/0/prometheus'
      - '-notifier.url=http://alertmanager1:9093'
      - '-notifier.url=http://alertmanager2:9093'
      - '-remoteWrite.url=http://vminsert:8480/insert/0/prometheus'
      - '-remoteRead.url=http://vmselect:8481/select/0/prometheus'
      - '-dedup.minScrapeInterval=30s'
    volumeMounts:
      - name: rules
        mountPath: /rules
volumes:
  - name: rules
    configMap:
      name: vmalert-rules-config
```

小結

- 多個 Alertmanager 可直接多次設 -notifier.url，提升通知高可用。
- VictoriaMetrics 與 Alertmanager 都有 dedup 機制，防止多份重複資料/通知。
- 設定 -remoteWrite.url/-remoteRead.url 可讓警報狀態持久化、重啟不丟失。
- vmalert 建議以 Deployment 部署於 K8s，replicas 調高即可 HA，所有 Pod 共用同一份 rule 設定。

如需進階案例或遇到 YAML、參數問題歡迎再問！



參考valert-ha.pdf，以繁體中文詳細說明參數：

apiVersion: apps/v1

kind: Deployment

metadata:

name: valert-ha

spec:

replicas: 2 # 兩個 Pod，達到 HA

selector:

matchLabels:

app: valert

template:

metadata:

labels:

app: valert

spec:

containers:

- name: valert

image: victoriametrics/valert:v1.95.2

args:

- '-rule=/rules/rules.yml'

- '-'

datasource.url=http://vmselect:8481/select/0/prometheus'

- '-notifier.url=http://alertmanager:9093'

- '-'

remoteWrite.url=http://vminsert:8480/insert/0/prometheus'

- '-'

remoteRead.url=http://vmselect:8481/select/0/prometheus'

- '-dedup.minScrapeInterval=30s' # 根據 rule

group interval 設定

volumeMounts:

- name: rules

mountPath: /rules

volumes:

- name: rules

configMap:

```
name: vmalet-rules-config # 預設 rules.yml 存  
於 ConfigMap
```



以下根據你貼的 Deployment YAML，以及 vmalet-ha.pdf 文件內容，以繁體中文詳細說明每個參數在高可用部署中的作用：

1. `apiVersion: apps/v1`

Kubernetes 的 API 版本，`apps/v1` 是現行建議用於 Deployment、StatefulSet 等控制器的版本。

2. `kind: Deployment`

Kubernetes 物件類型，這裡是 Deployment，代表會根據你設定的 replicas 數自動管理無狀態 Pod 的數量，適合 vmalet 這種 stateless（無狀態）服務。

3. `metadata: name: vmalet-ha`

物件名稱，K8s 內部辨識這個 Deployment 的名字，建議用容易理解的名稱，如 `vmalet-ha`（HA = 高可用）。

4. `spec: replicas: 2`

- **replicas** 指定要同時運作幾個 Pod（這裡設 2），用來實現 vmaalert 的 HA（高可用），避免單點故障。
-

5. `selector` 與 template.metadata.labels``

- `selector/matchLabels` 以及 `template.metadata.labels` 設為相同值（`app: vmaalert``），用來告訴 K8s 這個 Deployment 控制哪些 Pod（標籤要完全一致）。
-

6. `containers:``

K8s Pod 內運行的容器配置，下面各參數分別說明：

```
`- name: vmaalert`
```

容器名稱。

```
`- image: victoriametrics/vmaalert:v1.95.2`
```

vmaalert 的映像檔與版本（這裡指定 `v1.95.2``），建議固定版本以避免自動升級造成潛在問題。

```
`- args:`
```

這裡是 vmaalert 容器的啟動參數，以下每一個都非常關鍵：

6.1 ``- '-rule=/rules/rules.yml'``

- 指定規則檔路徑，vmaalert 會依此讀取 Prometheus 格式的警報規則。
 - ``/rules/rules.yml`` 是 Pod 內部的路徑，實際來源是 ConfigMap（下方 volumeMounts 對應）。
-

6.2 `--datasource.url=http://vmselect:8481/select/0/prometheus`

- 設定要查詢的資料源 URL。
 - 通常指向 VictoriaMetrics 的 vmselect，代表所有 vmalet 都去這個 endpoint 取得即時指標數據，支援 HA 結構（多個 vmalet 可指向同一 vmselect）。
-

6.3 `--notifier.url=http://alertmanager:9093`

- 指定要發送警報的 Alertmanager 位置。
 - 若有多組 Alertmanager HA，可多次寫這個參數，vmalet 會同時通知所有 Alertmanager。
-

6.4 `--remoteWrite.url=http://vminsert:8480/insert/0/prometheus`

- 警報狀態與 recording rule 結果會持久化寫入 VictoriaMetrics（cluster 需指 vminsert），讓 HA 下多個 vmalet 共享狀態。
 - 這樣即使 pod 重啟或輪替，警報 firing 狀態仍會被復原，資料不會遺失。
-

6.5 `--remoteRead.url=http://vmselect:8481/select/0/prometheus`

- vmalet 啟動時會自動從這個 URL 讀回 alert/recording 狀態。
 - 通常跟 datasource.url 設一樣，cluster 架構指向 vmselect。
-

6.6 `--dedup.minScrapeInterval=30s`

- 啟用 VictoriaMetrics 的**去重（deduplication）**機制，避免多個 vmaalert HA 實例產生的數據重複寫入。
 - 這個值應與你所有警報 rule group 設定的 `interval` 相同或其最大公因數，這樣才可正確去重。
 - 例如 rules group interval 為 30s/60s，就設成 30s 或 60s。
-

7. `volumeMounts` 及 `volumes`

7.1 `volumeMounts`

- 將 Pod 內的 `/rules` 資料夾對應到底下的 rules volume（實際上是掛載 ConfigMap）。

7.2 `volumes`

- `configMap: name: vmaalert-rules-config`
 - 表示 rules.yml 檔案來自 ConfigMap，這樣多個 Pod 可以共用同一份警報規則，也方便日後熱更新規則（例如直接 `kubectl apply -f` 更新 ConfigMap 即可）。
-

8. 部署建議

- 建議 ConfigMap 也 version control，與 pipeline 整合。
 - replicas 可視流量與高可用需求調整。
 - 多個 Alertmanager 時，記得每個都寫 `-notifier.url`，確报警報不會因單一點失效。
-

總結

這份 YAML 代表 **vmalert** 高可用正確部署方式，包含：

- 兩個 vmalert Pod (replicas: 2)
- 共用同一份警報規則 (ConfigMap)
- 多個實例查詢同一 VM 數據源，寫入同一 vminsert
- 去重與狀態持久化設定完整

如需 YAML 熱更新、ConfigMap 管理、更多 Alertmanager HA 或 dedup 進階參數說明，歡迎再詢問！



參考 [vmalert-ha.pdf](#)，以繁體中文詳細說明: Alertmanager HA 或 dedup 進階參數



根據 **vmalert-ha.pdf** 以及官方文件，以下用**繁體中文**詳細說明「Alertmanager HA」與「dedup 進階參數」兩大主題，並補充進階實務要點：

一、Alertmanager HA（高可用）

1. 什麼是 Alertmanager HA？

Alertmanager HA 就是同時部署多個 Alertmanager 節點，彼此協同工作。即使部分 Alertmanager 節點失效，警報通知與消警（resolved）功能都**不中斷**，保障監控告警系統的可靠性。

2. 運作原理

- 多個 Alertmanager 節點會互相同步狀態（使用 mesh network/cluster 機制）。
- vmalert 可以同時向多個 Alertmanager 發送警報，只需設定多個 `-notifier.url` 參數（vmalert-ha.pdf 明確說明此用法）。
- 每個 Alertmanager 收到告警後，會根據告警的 **labels（標籤）** 進行「去重」（deduplication），同一告警（label set 完全相同）只會發送一次。

3. Alertmanager HA 部署重點

- 建議多個 Alertmanager 彼此組成 HA mesh（[官方參考](#)）。
- vmalert 的參數可如下設置（支援多個 Alertmanager）：

```
-notifier.url=http://alertmanager1:9093  
-notifier.url=http://alertmanager2:9093
```

- vmalert 會將每個觸發的警報，同步送到所有 Alertmanager。
- 只要有一個 Alertmanager 正常運作，通知就不會漏掉。

4. 進階實務建議

- 建議使用 headless Service 或 LoadBalancer 方式讓多個 Alertmanager 之間能互相發現。
- 需確保每個 Alertmanager 配置檔有正確的 cluster/mesh 設定（參考 [alertmanager.yml](#) 的 cluster 部分）。
- 支援多租戶時，建議以 labels 加強警報隔離。

二、Dedup 進階參數（vmalert 及 VictoriaMetrics）

1. dedup（去重）原理

- 多個 vmalet HA 實例同時執行相同規則，產生相同 label set 的時序數據（警報、recording rule）。
- **VictoriaMetrics** 會根據 label set 與 timestamp 做 deduplication，避免同一數據重複寫入，確保只保留一份。

2. 關鍵 dedup 參數說明

(1) ``-dedup.minScrapeInterval``

- 這是 **vmalet** 的關鍵 **dedup** 參數，決定去重時的最小資料間隔。
- 設定原則：**需等於你 rules group interval 或最大公因數**，這樣 VictoriaMetrics 才能正確分辨同一批資料來自多個 vmalet。
- **範例**：若規則 group interval 設 30s，則 ``-dedup.minScrapeInterval=30s``
- 如果不同 rules group interval 不同，請設最大公因數（如 30s、60s 時，用 30s）

(2) ``-external.label``

- 為每個 vmalet Pod 指定獨特 external label（如 replica=pod-name），方便日後查詢與 debug。
- 如果你想更容易分辨來源，也可以這樣設（Pod name 通常用 Downward API 帶入）：

```
- '-external.label=replica=$(POD_NAME)'
```

(3) ``-evaluationInterval``

- 決定規則的全域計算頻率，會影響 dedup 機制的同步性。
- 建議和 group interval 配合，所有 vmalet 必須保持一致。

(4) ``-remoteWrite.url`` 與 ``-remoteRead.url``

- 前者負責寫入警報與 **recording rule** 結果，後者負責重啟時恢復警報狀態，確保多 vmalet HA 可同步狀態與資料。

3. 進階 dedup 配置建議

- 多個 vmalet 必須用完全相同的 **rules** 與 **labels**，才能讓 dedup 機制正確工作。
- 若 rules 有不同的 group interval，需小心設 dedup.minScrapeInterval，防止部分資料重複，部分被去掉。
- 維持規則間隔（interval）穩定，避免因 HA Pod 同步不齊導致誤判。

三、總結

- Alertmanager HA：部署多個 Alertmanager，vmalet 可同時發送警報到多個 `-notifier.url``，保證通知不中斷。
- Dedup 進階參數：`-dedup.minScrapeInterval`` 與 rules interval 配合，`-external.label`` 方便監控來源，`-remoteWrite.url`/-remoteRead.url`` 持久化狀態，多個 vmalet 實例用同一份 rules 與 labels。
- 這些設計確保了整體警報系統的高可用與數據一致性，並防止重複資料或通知。

如需範例 Alertmanager HA yaml 或 vmalet dedup 配置細節，可再提出！



參考vmalet-ha.pdf，以繁體中文詳細說明：
如果Vmalet 連線不到vmagent，但vmselect裡有
vmagent，這樣是不是還是能監控，當vmagent的cpu使用率
超過10%時，發出vmalet呢



這是個很實務且常見的狀況，以下參考 [vmalert-ha.pdf](#) 與 **VictoriaMetrics** 架構，以繁體中文詳細說明：

一、VictoriaMetrics 架構回顧

- **vmagent**：負責從各種 exporter 拉 metrics，並轉發給 vminsert（寫入）或 VictoriaMetrics 單機（推送）。
 - **vminsert/vmselect/vmstorage**：構成 VictoriaMetrics 叢集的資料流動主軸。
 - **vminsert**：接收寫入流量
 - **vmselect**：負責查詢（提供 PromQL/HTTP API 給 vmalert、Grafana 等）
 - **vmalert**：查詢 vmselect，根據規則計算是否要發警報。
 - **vmselect**：「查詢接口」，會統一聚合所有 vmagent 傳過來的時序資料，只要資料已經被寫入到 VictoriaMetrics 的資料庫，**vmselect** 就查得到。
-

二、vmalert 的資料流向與查詢邏輯

- **vmalert** 並不直接連線 vmagent。
 - 它只會根據 `-datasource.url`（通常是 vmselect）進行查詢，評估 PromQL/規則。
 - vmagent 只要有把 metrics 成功轉送到 vminsert（或單機 VictoriaMetrics），**vmselect** 查得到，**vmalert** 就能監控。
-

三、實際情境說明

1. 假設情境

- 你的 vmagent 斷線，**vmalert** 直接連不到 **vmagent**。
- 但是，**vmselect** 仍然聚合並保存了 **vmagent**（或被 **vmagent** 轉發過來的 **exporter**）所產生的資料。

2. 結果？

- 只要 vmagent 原本拉取的 metrics 有被寫進 VictoriaMetrics（無論是 cluster 或單機），
- vmalert 就能透過 vmselect 查詢到最新資料並判斷警報。
- 你依然可以監控 **vmagent** 的 **CPU** 使用率，並根據規則發出警報。

3. 警報條件範例

假設你有如下 rule，針對 vmagent job 做 cpu 監控：

```
groups:
- name: vmagent_cpu_alert
  rules:
  - alert: VmagentHighCPU
    expr: rate(process_cpu_seconds_total{job="vmagent"}[5m]) * 100 > 10
    for: 1m
    labels:
      severity: critical
    annotations:
      summary: "vmagent CPU 使用率過高"
```

“vmalert 查詢 vmselect，如果 metrics 還有新資料（即 vmagent 有傳資料進 VM），警報就會正常運作。”

四、例外狀況與注意事項

1. 前提：vmagent 必須有資料持續進到 VictoriaMetrics !

- 如果 vmagent 本身掛掉或不能把資料送到 vminsert/vmstorage，則 vmalert 查到的 metrics 會停留在舊資料或消失。

2. 若 vmagent 長時間沒資料進來，警報會自動失效或恢復

- Prometheus/VictoriaMetrics 規則的特性是：**expr** 查不到資料會視為未達到條件，所以警報會「消失」或不再觸發。
- 建議針對資料中斷可再設一個「資料消失」的警報：

```
- alert: VmagentNoData
  expr: absent(process_cpu_seconds_total{job="vmagent"})
  for: 5m
  labels:
    severity: warning
  annotations:
    summary: "vmagent 5分鐘未上報資料"
```

3. 結論

- vmalert 完全不需要直接連 vmagent，只要能查到 vmagent 產生的 metrics，就能發警報。
- 你的警報查詢都是對 vmselect（聚合所有歷史與即時數據）。

五、總結

- 只要 vmselect 查得到 vmagent 的 metrics，vmalert 就能照常監控與發警報。
- vmalert 不需與 vmagent 有直接連線，只要資料寫進 VictoriaMetrics，監控/警報都正常。
- 若 vmagent 掛掉或資料未寫入，則警報查不到資料，會判斷為未達警戒（或用 absent 類型額外設警報補強）。

如需補充警報語法、如何設 absent-type 告警，或進一步設計 vmagent 健康監控，歡迎再詢問！