

Prometheus 加密設定



以繁體中文詳細說明prometheus scrape job要加上密碼



在 Prometheus 中，若要對某個 scrape job 加上密碼（通常是 Basic Authentication），可以透過 ``basic_auth`` 設定來達成。這種方式常見於你的目標服務（exporter）需要身份驗證的情境。

以下是詳細說明：



一、Basic Auth 是什麼？

Basic Authentication（基本身份驗證）是 HTTP 協議的一部分，會將使用者名稱與密碼透過 Base64 編碼後，附加在每次的 HTTP 請求中。

格式為：

```
Authorization: Basic base64(username:password)
```



二、Prometheus 如何設定 Basic Auth？

在 Prometheus 的 ``prometheus.yml`` 中，``scrape_configs`` 的每個 job 可以使用 ``basic_auth`` 欄位來加入使用者名稱與密碼。



範例設定：

```
scrape_configs:
  - job_name: 'my_secure_exporter'
    static_configs:
```



```
- targets: ['localhost:9100']
basic_auth:
  username: 'myuser'
  password: 'mypassword'
```

這段設定會讓 Prometheus 在向 `localhost:9100` 發送 scrape 請求時，自動加上 Basic Auth 頭部。

✅ 三、如果密碼比較敏感，如何避免寫死在設定檔中？

Prometheus 支援使用檔案讀取方式設定密碼，這樣可以避免直接把密碼寫在 `prometheus.yml` 裡。

🔒 使用 `password_file`：

```
scrape_configs:
- job_name: 'my_secure_exporter'
  static_configs:
    - targets: ['localhost:9100']
  basic_auth:
    username: 'myuser'
    password_file: '/etc/prometheus/secrets/my_password.txt'
```

這裡的 `/etc/prometheus/secrets/my_password.txt` 是一個純文字檔案，內容只包含一行密碼。

“✅ 適用於密碼存放在 K8s Secret、CI/CD 管理或檔案掛載的情境。”

🔒 四、Exporter 端也要支援 Basic Auth

要注意的是，你的目標 exporter 也要能處理 Basic Auth 驗證。舉例來說：

- 若 exporter 是自己寫的 Python server，就要從 `Authorization` header 解析 Basic Auth。
- 若使用 nginx 當反向代理，也可以在 nginx 中加上 Basic Auth。

五、進階建議

若需要更高的安全性，也可以考慮：

- 使用 HTTPS + Basic Auth。
- 使用 bearer token (`authorization` header)。
- 將 Prometheus 部署在內部網路，限制對外存取。

需要我提供 nginx 設定 Basic Auth 的範例或 K8s Secret 整合方式嗎？可以一起補上。



Prometheus Remote Secrets.p...
PDF

參考"Prometheus Remote Secrets.pdf"，以繁體中文詳細說明



根據你提供的《**Prometheus Remote Secrets**》文件，這份提案是為了解讓 **Prometheus** 能夠從遠端 **Secret Providers**（例如 **Kubernetes Secrets**）讀取密碼或其他機密資訊，來取代傳統將密碼寫死在 `prometheus.yml` 或放在本機檔案系統的做法。以下是完整的說明與重點整理，並附上實作範例：



為什麼 Prometheus Scrape Job 要用 Remote Secrets 加上密碼？

傳統方法的問題：

目前 Prometheus 支援以下兩種方式設定密碼：

```
password: <明文密碼>           # 安全性差
password_file: <檔案路徑>      # 需要掛載 Secret 檔案
```

缺點包括：

- 密碼需要硬編碼或掛載到本機檔案 → 不安全。
- 在 Kubernetes 中會增加部署與 Secret 掛載的複雜度。
- 若 Prometheus 被部署於多節點上，其他 Pod 可能存取這些檔案（Node-level 權限風險）。

Remote Secrets 解法設計（未來功能，正在提案中）

Prometheus 引入新的 `password_ref` 機制，搭配 `secret_providers` 區塊來實現機密讀取的動態化與集中管理。

設定結構總覽：

設定 `secret_providers`

這部分定義從哪裡取得密碼：

```
secret_providers:
- kubernetes_sp_config:
    config:
```

```
api-server: https://my-k8s-api
secrets:
  - name: xyz
    config:
      secret:
        name: my-k8s-secret
        namespace: my-namespace
        key: my-key
```

2 Scrape job 使用 `password_ref`

然後你就可以在 `scrape_configs` 中使用：

```
scrape_configs:
  - job_name: 'secure_job'
    static_configs:
      - targets: ['my-exporter:9100']
    basic_auth:
      username: 'myuser'
      password_ref: xyz
```

🔄 Prometheus 在每次 scrape 時，會根據 `password_ref` 查詢 secret provider 中對應名稱的密碼。

✨ 好處與特點

特點	說明
✅ 安全性提升	密碼不再存在於本機硬碟、ConfigMap 或 YAML 內。
✅ 易於管理	同一個密碼可重複使用於多個 job，不需要修改多份設定。
✅ 支援動態更新	Secret providers 可以實作快取與更新機制（例如用 Kubernetes Watch）。
✅ 多平台支援	可以擴充支援 AWS Secrets Manager、Vault、K8s Secrets...等

補充技術細節

- 每個 ``secret_provider`` 會實作 ``Fetch(ctx)`` 方法，於每次 scrape 時擷取密碼內容。
- 若是頻繁請求的平台（如 K8s），建議使用快取來減少 API server 壓力。
- Prometheus 會支援 Metrics 來監控 Secret 載入狀態（例如失敗次數、目前載入數量等）。

與未來方向

此提案目前仍在設計中，將來可能會：

-  支援更多 Secret Provider（如 AWS Secrets Manager、HashiCorp Vault）。
-  將原本的 ``password`` / ``password_file`` 轉換成新的 ``inline_sp_config``、``filesystem_sp_config``。
-  加入 Secret Federation 機制（集中存取與快取），避免多個 Prometheus 個體重複讀取 Secret。

小結與建議

若你目前：

建議

使用本機 ``password_file``

可繼續使用，但安全性有限

在 Kubernetes 環境中

建議規劃導入 Remote Secrets 機制，搭配 ``password_ref``

密碼管理由 Vault / AWS Secret 控制

建議持續關注此提案，將來可整合自動取得密碼的能力

如果你想要我幫你撰寫一份實際範例（包含 Kubernetes Secret + Prometheus 設定 YAML），也可以提出，我可以幫你搭好基本架構。需要嗎？



把Prometheus Remote Secrets.pdf翻成繁體中文



Prometheus Remote Secrets



Prometheus Remote Secrets

Prometheus 遠端機密 (Remote Secrets)

作者：Daniel Hrabovcak 文件狀態：審閱中 建立時間：2023 年 10 月 26 日 上次重大更新：2023 年 11 月 3 日

目標 允許 Prometheus 從遠端機密提供者 (Secret Providers) 讀取機密資料。

提案 引入類似服務發現 (Service Discovery) 的機密發現機制，允許不同的機密提供者撰寫相應的程式碼，以從其 API 中讀取機密。

我已將《Prometheus Remote Secrets.pdf》完整翻譯為繁體中文，並整理為文件「**Prometheus Remote Secrets**」，你可以在右側畫布中查看與編輯。

如果你希望我幫你進一步補充 Kubernetes Secret 實作範例、或整合進 GitLab CI/CD 或 Helm 等工具，也可以告訴我，我可以繼續協助。需要嗎？



以繁體中文詳細說明，我的Prometheus運行在虛擬機，目前是讓user自己修改gitlab pipeline來產生scrape job的yml，gitlab上不能有密碼要怎麼處理



你提到 Prometheus 是部署在 **虛擬機 (VM)** 上運行，目前的做法是使用者透過修改 **GitLab CI pipeline** 來產生 ``scrape_configs`` 的 YAML 檔案。
但由於 **安全性考量**，**GitLab** 上不能出現明碼密碼 (**Plaintext Password**)，這是非常合理的限制。

下面我會詳細說明如何處理這個問題，讓密碼不出現在 GitLab 上，同時又能讓 Prometheus 正確 scrape 需要密碼驗證的 exporter：

✅ 目標與限制

- ✅ **GitLab CI/CD** 上不能放密碼
- ✅ 產生的 **Prometheus** 設定 **YAML** 檔必須包含 ``scrape_configs``
- ✅ **Prometheus** 運行在 **VM** 上 (不是 **Kubernetes**)
- ✅ 密碼需要保密，不能寫在 **Git** 倉庫中

🔑 解決方案總覽

解法	說明	優點	缺點
1. 使用 <code>`password_file`</code>	密碼放在 VM 本機的檔案中	GitLab 無需存密碼	需要管理 VM 上的機密檔案

解法	說明	優點	缺點
2. 使用 Prometheus Remote Secrets 提案	支援從遠端 Secret Provider 取密碼	將來彈性高	尚未正式支援，需自行擴充
3. 使用 GitLab CI/CD Secret + Template 合併	GitLab 將密碼以 CI Secret 方式存放，不輸出到 YAML	CI 保密密碼	需特別處理 YAML 合併
4. 將密碼填入 YAML 後直接部署，不 commit	生成 `scrape.yml` 時從 Secret 填入	符合保密需求	無法追蹤生成內容、易錯誤

✅ 建議實作方式：GitLab Secret + VM password_file

這是最穩定且簡單的做法，步驟如下：



Step 1：在 GitLab 設定 Secret 變數

前往 GitLab 專案的：

“Settings → CI / CD → Variables”

新增：

- `SCRAPE_PASSWORD="my_secure_password"`（可遮蔽）
- `SCRAPE_USERNAME="prom_user"`（非必要，但也可保密）

🔧 Step 2：CI pipeline 自動產生 YAML，不含密碼



CI Job 不直接將密碼寫進 YAML，而是將 username/password 分離，例如這樣產生：

```
# scrape_config.tpl.yaml (不包含密碼)
scrape_configs:
  - job_name: "secure_target"
    static_configs:
      - targets: ["10.0.0.123:8080"]
    basic_auth:
      username: ${SCRAPE_USERNAME}
      password_file: "/etc/prometheus/secrets/my_password.txt"
```

在 GitLab CI 階段執行時：

- 這份 `scrape_config.tpl.yaml` 會被模板引擎處理（如 `envsubst` 或 `jinja2`）。
- 密碼則寫入目標 VM 的檔案 `/etc/prometheus/secrets/my_password.txt`，這檔案不經過 GitLab。

Step 3：在 Prometheus VM 上維護密碼檔案

於 VM 上創建：

```
sudo mkdir -p /etc/prometheus/secrets
echo "$SCRAPE_PASSWORD" > /etc/prometheus/secrets/my_password.txt
chmod 600 /etc/prometheus/secrets/my_password.txt
```

可手動建立，也可以：

- 使用 Ansible / Puppet / Shell script 自動部署。
- 或在 CI/CD 中 SCP 上傳密碼檔案（不推薦長期用，較難控管變更）。

⚙️ Prometheus 設定示範：

```
scrape_configs:
  - job_name: 'secure_target'
    static_configs:
      - targets: ['10.0.0.123:8080']
    basic_auth:
      username: prom_user
      password_file: /etc/prometheus/secrets/my_password.txt
```

🔒 額外安全強化建議

1. VM 上的密碼檔權限設為 600，僅允許 `prometheus` 用戶讀取。
2. 若密碼變更頻率高，可設計 script 定期從 GitLab Secret 抓取並更新。
3. 使用 CI/CD secret masking，確保密碼不會輸出在 job log 中。

✅ 延伸備案（進階）

若你願意實作 Remote Secrets 架構（如你提到的 PDF 提案），也可以考慮：

- 建立一個簡單的 **Secrets API Server**，讓 Prometheus 在 scrape 前透過 `password_ref` 查詢密碼。
- Prometheus 在 VM 上部署時改為定期查詢這個 Secrets server。
- 此方法可擴展到多個 job、多組密碼共用管理。

但這做法較複雜，建議在未來導入 Kubernetes 時再推進。

總結

你目前使用 GitLab CI pipeline 產生 Prometheus scrape job，但不能寫入密碼的情況下，最穩定且推薦的做法是：

 使用 ``password_file`` 搭配 **GitLab Secret** 與 **VM 本機儲存密碼檔案**。
這樣可以保持 GitLab pipeline 無密碼、部署安全又符合 Prometheus 原生支援方式。

如果你願意，我可以幫你寫一份 GitLab CI Job、template YAML 範本，還有 VM 安裝腳本範例，要嗎？



以繁體中文詳細說明: 使用 GitLab CI/CD Secret + Template 合併



好的，以下是「**使用 GitLab CI/CD Secret + Template 合併**」的詳細說明，這種方式可以讓你：

-  在 GitLab 中安全儲存密碼（不暴露在 YAML 檔或 Git 倉庫中）
-  自動產生包含密碼的 Prometheus ``scrape_configs`` 設定
-  不將任何密碼寫入 Git 倉庫或 commit 中

核心概念

將 Prometheus 的 ``scrape_config.yml`` 拆成「**模板檔**」+「**GitLab Secret 變數**」，在 CI pipeline 中動態合併成完整的設定檔，再部署至 Prometheus VM 上。

一、設定 GitLab CI Secret 變數

前往 GitLab 專案：

“Settings → CI / CD → Variables”

新增以下變數（請勾選「Mask」與「Protect」）：

名稱	值範例	用途
<code>`SCRAPE_USERNAME`</code>	<code>`myuser`</code>	Exporter 登入帳號
<code>`SCRAPE_PASSWORD`</code>	<code>`supersecret123`</code>	Exporter 密碼

二、撰寫 Prometheus 設定模板

建立一個 ``scrape_config.tpl.yml`` 檔案（不要 **commit** 生成後的正式檔案！）

```
# scrape_config.tpl.yml
scrape_configs:
  - job_name: "secure_exporter"
    static_configs:
      - targets: ["10.0.0.123:8080"]
    basic_auth:
      username: "${SCRAPE_USERNAME}"
      password: "${SCRAPE_PASSWORD}"
```

這是帶有環境變數的 YAML 範本，GitLab CI 在部署時會替換這些變數為實際值。

🔧 三、撰寫 GitLab CI Job 合併模板

```
# .gitlab-ci.yml
stages:
  - generate
  - deploy

generate_prometheus_config:
  stage: generate
  image: alpine
  script:
    - apk add --no-cache gettext
    - envsubst < scrape_config.tpl.yml > scrape_config.yml
  artifacts:
    paths:
      - scrape_config.yml
  only:
    - main

deploy_to_vm:
  stage: deploy
  image: alpine
  script:
    - echo "將 scrape_config.yml 傳送到 VM"
    - scp scrape_config.yml prometheus@your-vm-
ip:/etc/prometheus/scrape_config.yml
  only:
    - main
  dependencies:
    - generate_prometheus_config
```

✨ 說明：

- 使用 `envsubst` 將模板中的 `\${SCRAPE_USERNAME}`、`\${SCRAPE_PASSWORD}` 替換成 GitLab Secret 值。
- 用 `artifacts` 保留產生的 `scrape_config.yml`。

- 第二階段將設定檔上傳到 VM 中 Prometheus 的指定位置（可用 SCP / Ansible / Rsync）。

四、Prometheus 主設定引用外部檔案（選用）

若想將 scrape job 拆開，主設定檔可以這樣寫：

```
# prometheus.yml
scrape_configs:
  - job_name: 'default_metrics'
    static_configs:
      - targets: ['localhost:9090']

# 引用外部
rule_files:
  - /etc/prometheus/scrape_config.yml
```

“這樣可以讓 CI pipeline 只產生 scrape job，主設定維持穩定。”

五、注意事項與安全強化

項目	說明
✔ Secret 不會出現在 Git 倉庫	模板不含密碼，合併後檔案也不 commit
✔ 密碼不會出現在 Job Log	請在變數設定中啟用「Mask」功能
⛔ 不要將產出的 `scrape_config.yml` 加入版本控制	僅由 CI 動態產生與部署即可
✔ 限制 Job 執行權限	Protect CI Job，避免開發分支誤觸

項目	說明
✅ 檢查部署後權限	VM 上的設定檔須限制為 Prometheus 可讀（如 <code>`chmod 640`</code> ）

✅ 適用情境

- Prometheus **不在 Kubernetes 上**，而是獨立 VM。
- 你需要使用 CI/CD 產生帶有機密的設定，但 **又不能把密碼寫進 Git 倉庫**。
- 使用者只需要修改 exporter 設定，不需要碰密碼細節。

📌 小結

透過「GitLab CI Secret + Template 合併」的方式，你可以：

- ✅ 安全地使用密碼
- ✅ 自動生成 scrape config
- ✅ 不將敏感資訊留在 Git 倉庫
- ✅ 彈性支援 VM 環境部署

若你需要我幫你產出實際可用的 ``.gitlab-ci.yml``、範例 `tpl.yml``、VM 部署腳本，也可以再告訴我，我可以幫你整理一套範本 🚀





當你在使用 GitLab CI/CD、Docker、或 shell script 自動化產生配置檔時，``envsubst`` 是一個非常實用的工具，它可以幫助你把文字中的環境變數動態替換成實際值。

以下是完整的繁體中文說明：

什麼是 ``envsubst`` ？

``envsubst`` 是 GNU gettext 工具集中的一個小工具，作用是：

“將文字檔中的 ``${變數}`` 或 ``$變數`` 替換成環境變數的值。”

非常適合用來生成帶有敏感資訊（如密碼、Token）的設定檔，不需要把密碼寫在 Git 裡。

基本用法

假設你有一個模板檔案 ``config.tpl.yml``：

```
api_key: ${API_KEY}
user: ${USERNAME}
```

你先設定環境變數：

```
export API_KEY=abcdef123
export USERNAME=admin
```

然後使用 `envsubst`：

```
envsubst < config.tpl.yml > config.yml
```

結果會是：

```
api_key: abcdef123
user: admin
```

✓ 優點

特點	說明
✓ 簡單無依賴	不需要 Python、Jinja2、Node.js 等模板語言
✓ 支援 CI/CD	在 GitLab、GitHub Actions、Jenkins 都能直接用
✓ 保護機密	可搭配 GitLab Secret，把密碼動態填入但不寫死

🔑 範例：Prometheus `scrape_config.tpl.yml`

```
scrape_configs:
- job_name: secure_job
  static_configs:
    - targets: ["10.0.0.1:9100"]
  basic_auth:
    username: ${SCRAPE_USERNAME}
    password: ${SCRAPE_PASSWORD}
```

CI 設定：

```
# .gitlab-ci.yml
generate_scrape_config:
  image: alpine
  script:
    - apk add --no-cache gettext
    - envsubst < scrape_config.tpl.yml > scrape_config.yml
```

GitLab Secret：

- ``SCRAPE_USERNAME = prometheus``
- ``SCRAPE_PASSWORD = mysecret123``

最終產出 ``scrape_config.yml``，密碼不會出現在 Git 倉庫，也不會被 commit。

envsubst 進階用法

 指定變數範圍（只替換特定變數）：

```
envsubst '${SCRAPE_USERNAME} ${SCRAPE_PASSWORD}' < input.tpl > output.yml
```

只會替換這兩個變數，其它變數會保留。

！ 注意事項

問題

解法

變數沒有定義時會變空白

確保變數已 export，例如 ``export API_KEY=...``

問題	解法
不支援條件邏輯	<code>`envsubst`</code> 只做字串替換，不支援 if/else 或迴圈
YAML 中冒號、縮排要小心	保持格式正確，避免錯誤

小結

功能	說明
工具名稱	<code>`envsubst`</code>
功能	用環境變數取代文字檔中的變數
適用情境	CI/CD、自動化部署、模板產生
搭配環境	GitLab、Docker、Ansible、手動部署皆可

如果你需要我幫你寫一個完整的 ``envsubst` + GitLab CI` 範例專案結構，也可以說一聲，我可以幫你建立一整套開發流程範本！



如果 `scrape_config.tpl.yml` 有很多 放在不同某個資料夾呢？



如果你的 ``scrape_config.tpl.yml`` 有很多份，分散在某個資料夾中，例如：

```
scrape_jobs/  
├─ job_a.tpl.yml  
├─ job_b.tpl.yml  
└─ job_c.tpl.yml
```

你希望在 **GitLab CI/CD** 中自動把它們全部進行 `envsubst` 變數替換，並合併為一份完整的 Prometheus 設定或部署到某個目錄，以下是詳細的做法與建議。

目標

- 每個 job 的 `*.tpl.yml` 代表一個 `scrape_config`。
- 這些模板中含有 `\${變數}`，需透過 `envsubst` 替換。
- 最終要：
 - ☒ 將它們各自輸出成 `*.yml`；
 - ☒ 或合併成一份完整設定檔。

☒ 方法一：逐一套用 `envsubst`（保留為多檔）

檔案結構：

```
scrape_jobs/  
├─ job_a.tpl.yml  
├─ job_b.tpl.yml  
└─ job_c.tpl.yml
```

 `gitlab-ci.yml`

```
generate_scrape_configs:
  image: alpine
  stage: generate
  before_script:
    - apk add --no-cache gettext
  script:
    - mkdir -p rendered_jobs
    - for f in scrape_jobs/*.tpl.yml; do
        filename=$(basename "$f" .tpl.yml);
        envsubst < "$f" > "rendered_jobs/${filename}.yml";
      done
  artifacts:
    paths:
      - rendered_jobs/
```

這會把所有 `scrape_jobs/*.tpl.yml` 自動轉成：

```
rendered_jobs/
├─ job_a.yml
├─ job_b.yml
└─ job_c.yml
```

✅ 方法二：合併為單一 `scrape_config.yml`

如果你希望把所有 job 合併成一個檔案：

📦 `.gitlab-ci.yml` (合併)

```
generate_scrape_configs:
  image: alpine
  stage: generate
  before_script:
    - apk add --no-cache gettext
```

```

script:
  - echo "scrape_configs:" > scrape_config.yml
  - for f in scrape_jobs/*.tpl.yml; do
    echo "- " >> tmp_job.yml;
    envsubst < "$f" | sed 's/^/ /' >> tmp_job.yml;
  done
  - cat tmp_job.yml >> scrape_config.yml
artifacts:
  paths:
    - scrape_config.yml

```

合併後結果範例

```

scrape_configs:
  - job_name: "job_a"
    static_configs:
      - targets: ["a:9100"]
  - job_name: "job_b"
    static_configs:
      - targets: ["b:9100"]

```

提示：使用 `find` 處理子目錄也沒問題

若 `tpl` 檔不只在一層目錄，請用：

```
find scrape_jobs -name "*.tpl.yml"
```

搭配迴圈即可處理所有檔案：

```

find scrape_jobs -name "*.tpl.yml" | while read f; do
  ...
done

```

小技巧補充

忽略特定檔案

只處理命名以 ``secure_`` 開頭的 job：

```
for f in scrape_jobs/secure_*.tpl.yml; do ...
```

安全提醒

使用 ``envsubst`` 時：

- ``envsubst`` 替換後的密碼不應寫入 Git 倉庫。
- 可在部署階段後刪除輸出檔或將其存在 ``gitignore`` 控制外。

小結

需求	做法建議
每個 job 拆開保留	for 迴圈 + <code>`envsubst`</code> 個別輸出
合併為一份設定	用 shell 合併 + <code>`envsubst`</code> + <code>`sed`</code>
子資料夾掃描	用 <code>`find`</code> 遞迴搜尋
輸出檔案控制	用 GitLab <code>`artifacts`</code> 傳遞給後續步驟

如果你希望我幫你做一個 demo project 結構（例如完整 `.gitlab-ci.yml` + `scrape_jobs` 資料夾），我可以幫你整理一份 zip 包或模板專案，要嗎？



以繁體中文詳細說明，實作方式：GitLab Secret + VM password_file，並比較2者



好的，以下將以繁體中文詳細說明：

主題：實作方式：GitLab Secret + VM password_file

這種方式是 在 **GitLab CI/CD** 中透過 **Secret 變數管理機密**，並將密碼安全地寫入 VM 上的 `password_file`，再由 Prometheus 使用 `password_file` 存取該機密。
這是部署在 **VM 上的 Prometheus** 最常見、也最安全的密碼管理方法之一。

一、實作目標與流程

目標：

- 不將密碼寫入 Git 倉庫、YAML 檔或 GitLab commit
- Prometheus 在 scrape job 中透過 `password_file` 安全地讀取密碼
- 使用 GitLab CI/CD 將設定與密碼自動化部署到 VM

二、環境假設

條件	說明
Prometheus 部署	在 Linux 虛擬機 (VM)
密碼來源	GitLab Secret
使用方式	Prometheus scrape job 使用 <code>`password_file`</code>
部署方式	GitLab CI/CD + SCP / Ansible 等方式部署到 VM

三、實作步驟

Step 1：在 GitLab CI/CD 中建立 Secret

前往 GitLab → ``Settings`` → `CI / CD`` → `Variables``，加入下列變數：

變數名稱	說明	設定建議
<code>`SCRAPE_USERNAME`</code>	BasicAuth 帳號	可 Mask 或保留
<code>`SCRAPE_PASSWORD`</code>	BasicAuth 密碼	<input checked="" type="checkbox"/> Mask + <input checked="" type="checkbox"/> Protect

Step 2：Prometheus 設定範例 (`scrape_config.tpl.yml`)

這裡用 `.tpl.yml` 作為模板，讓密碼從檔案中讀取：

```
scrape_configs:
  - job_name: "secure_target"
    static_configs:
      - targets: ["10.0.0.123:9100"]
    basic_auth:
      username: "${SCRAPE_USERNAME}"
      password_file: "/etc/prometheus/secrets/${CI_JOB_NAME}_password.txt"
```

“`password_file` 指向 VM 本機的路徑。”

🚀 Step 3 : GitLab CI 實作 `.gitlab-ci.yml`

```
stages:
  - prepare
  - deploy

generate_scrape_config:
  stage: prepare
  image: alpine
  script:
    - apk add --no-cache gettext openssh
    - mkdir -p out
    - echo "${SCRAPE_PASSWORD}" > out/${CI_JOB_NAME}_password.txt
    - chmod 600 out/${CI_JOB_NAME}_password.txt
    - envsubst < scrape_config.tpl.yml > out/scrape_config.yml
  artifacts:
    paths:
      - out/

deploy_to_vm:
  stage: deploy
  image: alpine
  before_script:
```

```

- apk add --no-cache openssh
script:
- scp out/scrape_config.yml prometheus@your-
vm:/etc/prometheus/scrape_config.yml
- scp out/${CI_JOB_NAME}_password.txt prometheus@your-
vm:/etc/prometheus/secrets/${CI_JOB_NAME}_password.txt

```

📌 Step 4 : VM 權限設定（一次性）

在 Prometheus VM 上：

```

sudo mkdir -p /etc/prometheus/secrets
sudo chown -R prometheus:prometheus /etc/prometheus/secrets
sudo chmod -R 700 /etc/prometheus/secrets

```

🔍 四、與「GitLab Secret + Template 合併」方式比較

項目	使用 `password_file`	使用 `envsubst` 將密碼寫入 YAML
🔑 密碼存放方式	存在 VM 本機 `/etc/prometheus/secrets/*.txt`	寫入 YAML `password:` 欄位中
✅ GitLab 安全性	高，密碼不會出現在產出 YAML 中	中，需確保 YAML 不 commit、不輸出
🔄 CI/CD 流程彈性	密碼可與設定檔分開管理，支援多 job 重用	密碼會寫死在合併後的 config 中
👁 可讀性	Prometheus 設定中無密碼內容，較乾淨	Prometheus 設定檔直接包含密碼
🚨 安全風險	需確保 VM 上 `password_file` 權限安全	若 YAML 被人讀取，密碼可能外洩

項目	使用 <code>`password_file`</code>	使用 <code>`envsubst`</code> 將密碼寫入 YAML
 運用場景	適合正式環境 / 密碼頻繁更動情境	適合開發測試階段或快速部署

✅ 五、建議選用時機

使用場景	建議方式
密碼需要高安全性保護	✅ 使用 <code>`password_file`</code>
Prometheus 在 VM 上、無 Vault	✅ 使用 <code>`password_file`</code>
只是快速測試，環境封閉	可考慮 <code>`envsubst`</code> 合併
密碼未來考慮集中管理或自動更新	<code>`password_ref`</code> （未來）

小結

若你運行 Prometheus 在 VM，且希望 **安全地使用密碼**，又不想在 GitLab 中暴露敏感資訊，最佳方式是：

GitLab Secret + VM 上獨立的 `password_file` 配置

搭配 ``envsubst`` 動態生成 YAML 設定，部署到 VM，密碼與設定檔分離，實作簡單又安全。

需要我幫你寫一套完整的 GitLab CI + Prometheus 配置專案範本（包含 ``.gitlab-ci.yml``、VM 指令、樣板檔）嗎？我可以幫你快速整理好 



幫我寫一套完整的 GitLab CI + Prometheus 配置專案範本
(包含 .gitlab-ci.yml、VM 指令、樣板檔)



</> Gitlab Prometheus Ci



1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

在聊天中回答 >

我已幫你建立好完整的 GitLab CI + Prometheus 配置專案範本，現在你可以在右側畫布中看到
`.gitlab-ci.yml`。



專案結構建議：

```
prometheus-ci-project/  
├── .gitlab-ci.yml  
├── scrape_jobs/  
│   └── secure_app.tpl.yml  
└── README.md
```



範例 scrape job 模板：

`scrape_jobs/secure_app.tpl.yml`

```
scrape_configs:
  - job_name: "secure_app"
    static_configs:
      - targets: ["10.0.0.123:9100"]
    basic_auth:
      username: ${SCRAPE_USERNAME}
      password_file: "/etc/prometheus/secrets/${CI_JOB_NAME}_password.txt"
```

“這份模板會在 CI 階段被 `envsubst` 套用變數後輸出成真正的 Prometheus 設定。”



GitLab Secret 設定建議：

前往 GitLab 專案 → `Settings` → **CI / CD** → **Variables**，新增：

變數名稱

值

建議

`SCRAPE_USERNAME`

`prometheus`

可勾選 Mask ☒

`SCRAPE_PASSWORD`

`mysecret123!`

勾選 Mask ☒ Protect ☒



Prometheus VM 上的初始設定（僅一次）



```
sudo mkdir -p /etc/prometheus/secrets
sudo chown -R prometheus:prometheus /etc/prometheus/secrets
sudo chmod -R 700 /etc/prometheus/secrets
```

完整流程

1. 開發者修改 `secure_app.tpl.yml`，加入新的 target 或 job
2. **GitLab CI** 自動將模板轉換為正式設定並加入密碼
3. 自動 **SCP** 部署到 **VM** 的指定目錄
4. Prometheus reload 設定後即可生效

需要我幫你補上 README 說明檔或寫一段 Ansible 自動化腳本也可以，告訴我即可 