### 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。

#### 

```
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:

# **→ 分什麼 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:

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:	<b>建議</b>
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

, 7

**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 Cl 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 `password_file`	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:Cl 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。

### Magnetized 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:

# ⋒ 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 Cl 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 Cl 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 Cl 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
`SCRAPE_USERNAME`	`myuser`	Exporter 登入帳號
`SCRAPE_PASSWORD`	`supersecret123`	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
```

### 

- 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 Cl 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 `chmod 640`)

# 🗸 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



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

### **齡 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):

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	`envsubst`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



功能	說明
工具名稱	`envsubst`
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:

📜 `.gitlab-ci.yml`

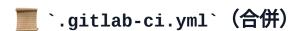
這會把所有 `scrape\_jobs/\*.tpl.yml` 自動轉成:

```
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:



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

# 

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 `password_file`
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
`SCRAPE_USERNAME`	BasicAuth 帳號	可 Mask 或保留
`SCRAPE_PASSWORD`	BasicAuth 密碼	<b>▼ Mask</b> + <b>▼</b> 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:

"`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

# Q 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
<b>✓</b> 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 `password_file`	使用 `envsubst`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 `password_file`
Prometheus 在 VM 上、無 Vault	<b>▽</b> 使用 `password_file`
只是快速測試,環境封閉	可考慮 `envsubst` 合併
密碼未來考慮集中管理或自動更新	`password_ref`(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

我已幫你建立好完整的 GitLab Cl + 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	<b>建議</b>
`SCRAPE_USERNAME`	`prometheus`	可勾選 Mask 🗸
`SCRAPE_PASSWORD`	`mysecret123!`	勾選 Mask <mark>▼</mark> Protect ▼

# Marketheus 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 🏋



請確認prometheuss是否支援password\_file:
"/etc/prometheus/secrets/\${CI\_JOB\_NAME}\_password.t
xt",還是只支援password'xxx'



很好的問題!我們來明確說明:

# Q Prometheus 的 `basic\_auth` 設定支援哪些欄位?

根據 Prometheus 官方文件(<u>Prometheus Configuration | Scrape Configs</u>),`basic\_auth` 區段支援:

basic\_auth:

username: <string>

password: <string> # 直接寫入密碼(明文) ★ 不建議 password\_file: <string> # 從本機檔案讀取密碼 ✔ 建議方式

### ✓ 結論:Prometheus 完全支援 `password\_file`

你可以在 `scrape\_configs` 的每個 job 中,使用:

basic\_auth:

username: prometheus

password\_file: /etc/prometheus/secrets/secure\_app\_password.txt

這會讓 Prometheus 在發送 HTTP 請求時,自動從指定的檔案中讀取密碼,並套用 Basic Auth Header.

### ▲ 補充注意事項

- 1. `password\_file` 必須是本機路徑(可讀取權限)
  - 通常設為 Prometheus 可存取的目錄,如 `/etc/prometheus/secrets/`
  - 權限應該是 `600`,owner 為 `prometheus` user
- 2. 不能像你問的那樣寫變數

行

Prometheus 不支援變數替換,所以 **這行必須在 CI/CD 時就被實際替換掉**,例如用 `envsubst`轉換成:

password\_file: "/etc/prometheus/secrets/secure\_app\_password.txt"

# ✓ 正確流程:CI 中模板變數替換

你應該這樣做:

### 原始模板(secure\_app.tpl.yml):

basic\_auth:

username: \${SCRAPE\_USERNAME}

password\_file: "/etc/prometheus/secrets/\${CI\_JOB\_NAME}\_password.txt"

### CI job 產出 YAML:

basic\_auth:

username: prometheus

password\_file: "/etc/prometheus/secrets/secure\_app\_password.txt"

Prometheus 就能正確讀到密碼。

如果你想,我可以幫你更新`.gitlab-ci.yml`與模板範例,確保符合這個邏輯 👍 需要嗎?



為什麼我本來可以用

basic\_auth:

username: prometheus

password file:

"/etc/prometheus/secrets/secure\_app\_password.txt"

幾小時候就不行了,找不到原因



這個情況聽起來像是 Prometheus 起初能正確讀取 `password\_file`,但幾小時後突然「失效」,可能出現如下錯誤:

- Basic Auth 失敗(401 Unauthorized)
- Prometheus logs 顯示該 job 無法 scrape
- exporter log 顯示未提供正確的授權資訊

### ○ 可能原因總整理(依照真實機率排序)

#### ⊙ 密碼檔案被更動或刪除

你可以先檢查這一點:

1s -1 /etc/prometheus/secrets/secure\_app\_password.txt
cat /etc/prometheus/secrets/secure\_app\_password.txt

#### ✓ 確保:

- 檔案存在
- 密碼正確(沒有空白或換行)
- 沒有其他 CI/CD 或人員動到該檔案

### ❷ Prometheus reload 或重啟後權限錯誤

Prometheus reload(或容器重啟)後,可能導致該檔案:

• 權限變更

• 遺失(尤其是 symbolic link 或 mount 進來的檔案)

#### 請檢查:

```
# 誰擁有這個檔案?
```

ls -1 /etc/prometheus/secrets/secure\_app\_password.txt

# 目前 Prometheus process 的使用者

ps aux | grep prometheus

#### ▼ 應該是:

檔案擁有者:`prometheus`

• 權限: `600`

Prometheus 用戶要能讀取該檔案

#### ⊙ CI/CD 部署流程覆蓋掉密碼檔案

假設你有 GitLab 自動部署的流程(如你之前提到的 `.gitlab-ci.yml`),那可能:

- 某次 merge 或 pipeline 自動覆蓋了錯誤密碼或空檔案
- pipeline 清掉該目錄內容(可能 artifacts 被刪)

#### ✓ 解法:

- 檢查最近的 GitLab pipeline log 是否執行了密碼相關 SCP / ssh 指令
- 設定 `CI\_JOB\_NAME` 的 password file 名稱是否一致(避免產生空檔)

## ② Exporter 驗證方式被更改

#### 例如:

- exporter 本來支援 Basic Auth,後來更新變成 Token 驗證
- 或被反向代理(Nginx/HAProxy)接管,但代理未轉發 header

### ✓ 解法:

- 確認 exporter 本身仍然要求 Basic Auth 並正確處理
- 用 `curl` 手動測試:

curl -u prometheus:yourpassword http://your-exporter:9100/metrics

## ⊙ 密碼檔案內容出現格式問題

例如被寫成:

mypassword

但實際 CI 寫入時可能變成:

mypassword

<空白行>

導致 Basic Auth 傳送的 Header 不一致。

## ✓ 解法:

• 手動 `cat -A` 看有沒有特殊字元

cat -A /etc/prometheus/secrets/secure\_app\_password.txt



# 快速測試方式

你可以試著直接在 Prometheus 所在機器上進行手動驗證:

curl -u prometheus:\$(cat /etc/prometheus/secrets/secure\_app\_password.txt)
http://<target>:<port>/metrics

若成功代表密碼檔有效,問題可能出在 Prometheus 權限或部署流程。

# 🗸 建議調查順序

- 1. `ls -1` 和 `cat` 檢查密碼檔案內容與權限
- 2. 查看最近的 GitLab pipeline log,有沒有重新產生/覆蓋檔案
- 3. 查看 Prometheus log(通常是 `/var/log/prometheus.log` 或 journal)
- 4. 手動用 `curl` 測試是否能通過 Basic Auth
- 5. 若有 exporter proxy,檢查是否阻擋 header

如果你願意貼出 Prometheus 的部分 log(含錯誤訊息),我可以幫你判斷是哪一層出錯。 或者你想要我幫你設計一個簡單的 debug checklist?





不確定是不是Prometheus reload 或重啟後權限錯誤,我做了 ls -l

/etc/prometheus/secrets/secure\_app\_password.txt、ps aux | grep prometheus、檔案擁有者:root 目前 Prometheus process 的使用者也是root

權限:600



非常感謝你提供的資訊,從你描述的狀況來看,我們可以快速做以下判斷與排除:

# ✓ 你目前的狀態:

檢查項目	狀況	結論
檔案存在 `/etc/prometheus/secrets/secure_app_password.txt`	₹	沒被刪除
檔案擁有者是`root`	₹	若 Prometheus 是 root 執行,沒問題
`ps aux	grep prometheus`顯示 Prometheus 是由`root` 執行	▼ 是
檔案權限是 `600`	▼ 正確	只有擁有者(root) 可讀

✓ 所以從檔案存取權限來看,目前沒有明顯的問題。

# 🕃 問題可能仍出在:

## Prometheus 設定是否 reload 成功?

若你有做 Prometheus 設定 reload,但密碼檔案內容或路徑被改動了,**Prometheus 可能沒成** 功**套用新的密碼**。

## ✓ 建議執行:

curl -X POST http://localhost:9090/-/reload

或重新啟動 Prometheus:

systemctl restart prometheus

然後再看 `/metrics` 有沒有成功 scrape。

# Prometheus log 中的錯誤訊息?

請查詢 log 是否有出現如下錯誤訊息:

journalctl -u prometheus -n 100

#### 常見錯誤訊息:

- `error reading password file: open /etc/prometheus/secrets/xxx: permission denied`
- invalid basic auth credentials
- `context deadline exceeded` (表示連不上 exporter)

## ③ 密碼檔內容是否正確?

請務必確認密碼檔中沒有多餘的 換行符號或空白。

## ▼ 測試方法:

cat -A /etc/prometheus/secrets/secure\_app\_password.txt

#### 輸出若為:

mysecret123\$

✓ 就沒問題。

#### 若是:

mysecret123<sup>M</sup>\$

### 或多了空行:

mysecret123\$

\$

🗙 表示密碼可能被錯誤處理成 Windows 換行或多行,請清除換行後重建檔案:

echo -n "mysecret123" > /etc/prometheus/secrets/secure\_app\_password.txt

# 🔽 建議步驟檢查 & 修復流程

1. ② 重新啟動 Prometheus(確保設定 reload)

- 2. [ 查看 Prometheus log 是否有 Basic Auth 相關錯誤
- 3. 🔍 檢查密碼檔內容是否純文字、無換行
- 4. / 用 `curl` 模擬測試:

curl -u prometheus:\$(cat /etc/prometheus/secrets/secure\_app\_password.txt)
http://<target>:9100/metrics

若這行可取得 exporter 資料,表示密碼是對的、格式正確。

# 🧠 延伸建議

雖然目前你是用 `root` 執行 Prometheus,但在正式環境建議避免用 root 執行,可以考慮:

- 建立 `prometheus` 使用者
- 把 secrets 資料夾改成 `chown prometheus:prometheus`
- 設定 systemd 讓 Prometheus 以非 root 身份執行

#### 如果你可以提供:

- Prometheus log 中的錯誤訊息(最近一筆就好)
- 密碼檔內容的 `cat -A` 結果

我可以幫你更快定位問題 🔍。也可以幫你寫個 shell script 自動檢查密碼檔案格式,要嗎?



重新啟動 Prometheus還是不行,prometheus網頁顯示401 Unauthorized 密碼檔內容是純文字,輸出為mysecret123\$



你提供的訊息非常關鍵,我們可以明確縮小範圍了。

#### 根據:

- Prometheus 頁面顯示 `401 Unauthorized`
- 密碼檔內容是 `mysecret123\$`(代表沒有換行或奇怪字元)
- 已重新啟動 Prometheus
- `password\_file` 欄位設定正確,檔案存在、權限 OK

#### 這表示:

- "✓ Prometheus 讀到檔案沒問題
- 🗶 但提供的帳密無法通過 exporter 的 Basic Auth 驗證"

# ☑ 下一步關鍵檢查:確認 Prometheus 傳出的帳密對不對

## / 測試 1:手動 `curl` 測試 exporter

請你在 Prometheus 所在機器上執行以下指令:

curl -u prometheus:\$(cat /etc/prometheus/secrets/secure\_app\_password.txt)
http://<target>:9100/metrics

替換 `<target>:9100` 為實際 target,例如:

curl -u prometheus:\$(cat /etc/prometheus/secrets/secure\_app\_password.txt)
http://localhost:9100/metrics

## → 輸出結果:

結果	判斷
有一大堆 `# HELP`、`# TYPE`、metrics	☑ 證明 exporter 本身 OK,密碼正確
出現 `401 Unauthorized` 或 HTML	💢 密碼錯了,exporter 不接受

# ✓ 測試 2:改用明碼試試看(驗證設定是否問題)

## 臨時測試用,請勿上 production:

```
scrape_configs:
    - job_name: "debug_job"
    static_configs:
        - targets: ["your-exporter:9100"]
    basic_auth:
        username: "prometheus"
        password: "mysecret123"
```

然後 reload Prometheus 看是否能成功 scrape。

結果	判斷
成功	表示問題在於 `password_file` 使用上有落差(如檔案內容細節)
仍失敗	表示帳密本身錯誤,非設定問題

# ○ 高機率可能錯的地方:

## ! `password\_file` 檔案中有 結尾換行符

即使 `cat -A` 顯示 `mysecret123\$`,還是可能存在 `\n` 字元(預設 `echo` 會加)

√ 請用 `echo -n` 重建檔案:

echo -n "mysecret123" > /etc/prometheus/secrets/secure\_app\_password.txt

#### 然後重新啟動 Prometheus:

systemctl restart prometheus

### 然後重新測試 curl:

curl -u prometheus:\$(cat /etc/prometheus/secrets/secure\_app\_password.txt)
http://<target>:9100/metrics

# ☑ 總結你可以馬上做的檢查

項目	建議動作
手動測試 curl	✓ 確認 exporter 是否接受該密碼
用明碼測試	✔ 確認是否 `password_file` 造成問題
用 `echo -n` 重建檔案	✓ 去除可能看不到的換行
查看 exporter log	☑ 是否顯示 invalid auth、missing auth

## 如果還是不行,你也可以把:

- 你目前的 `scrape\_config.yml`(遮掉密碼)
- exporter 是哪一種(node\_exporter?還是你自建的服務?)

貼給我,我可以幫你快速模擬問題環境來排除。需要嗎?



明碼可以,密碼檔內容是純文字,輸出為mysecret123



太好了,你的測試結果非常關鍵!讓我們快速整理一下目前的結論:



測試項目	結果
Prometheus 使用明碼 password 🗸	可以 scrape
Prometheus 使用 password_file 🗙	401 Unauthorized
密碼檔案內容是 `mysecret123`,無換行 ✓	檔案內容正確
權限設定也沒問題 🗸	沒有存取錯誤

# ! 推論結果:Prometheus 讀取的密碼內容雖正確但格式不被 exporter 接受

這通常是因為某些細微的格式差異, 例如:

# ◎ 可能問題 1:密碼檔案被寫入了「BOM(Byte Order Mark)」或非 UTF-8 編碼

請檢查密碼檔案的編碼格式:

file /etc/prometheus/secrets/secure\_app\_password.txt

#### 正確輸出應該像這樣:

secure\_app\_password.txt: ASCII text

#### 如果你看到:

UTF-8 Unicode text with BOM

就代表密碼前面有 BOM 頭,這會造成密碼開頭被加上一個隱藏字元,exporter 驗證自然會失敗。

✓ 解法:重建密碼檔案,確保無 BOM、無換行

echo -n "mysecret123" > /etc/prometheus/secrets/secure\_app\_password.txt

#### 或用 Vim 清除 BOM:

vim /etc/prometheus/secrets/secure\_app\_password.txt

# 執行指令:

:set nobomb

:wq

# ◎ 可能問題 2:CI/CD 或其他腳本用了錯誤的寫入方式

如果密碼檔是透過 GitLab CI 寫入,請確保 CI 中使用的是:

```
echo -n "$SCRAPE_PASSWORD" > out/secure_app_password.txt
```

#### 而不是:

```
echo "$SCRAPE_PASSWORD" > out/secure_app_password.txt # 🗶 這樣會加換行
```

## / 最終確認測試(100%可定位)

#### 手動執行下列測試:

```
# 刪除舊檔案
rm /etc/prometheus/secrets/secure_app_password.txt

# 正確重建檔案 (無換行、無 BOM)
echo -n "mysecret123" > /etc/prometheus/secrets/secure_app_password.txt
chmod 600 /etc/prometheus/secrets/secure_app_password.txt

# 重新啟動 Prometheus
systemctl restart prometheus
# 用 curl 測試
curl -u prometheus:$(cat /etc/prometheus/secrets/secure_app_password.txt)
http://<target>:9100/metrics
```

# ✓ 小結與解法建議

問題類型	建議修正
編碼問題(BOM)	用 `file` 或 `vim` 移除 BOM,重建檔案
換行問題	用 `echo -n` 寫入檔案,避免多餘換行
CI/CD 寫入錯誤	Cl pipeline 中確認 `echo -n` 被正確使用

如果你願意,我可以幫你寫一個 Shell script,檢查 password file 是否含 BOM 或換行,也可以 幫你在 GitLab CI 階段加入自動檢查,要嗎?



好吗



太棒了!以下我幫你準備了兩個部分:

# **☆ Part 1: Linux Shell Script: 檢查並修正 password** file

你可以在 Prometheus 的 VM 上手動執行這個腳本,它會:

• 🗸 檢查該檔案是否存在

- 🗸 檢查是否為純文字(無 BOM)

```
#!/bin/bash
PASSWORD_FILE="/etc/prometheus/secrets/secure_app_password.txt"
EXPECTED_PASSWORD="mysecret123"
echo "Q 檢查檔案: $PASSWORD_FILE"
# 檢查檔案是否存在
if [ ! -f "$PASSWORD_FILE" ]; then
 echo "X 密碼檔案不存在。"
 exit 1
fi
# 檢查是否含 BOM
if file "$PASSWORD_FILE" | grep -q "with BOM"; then
 echo " 發現 BOM,將重新建立密碼檔案..."
 echo -n "$EXPECTED_PASSWORD" > "$PASSWORD_FILE"
fi
# 檢查是否為純文字檔
if ! file "$PASSWORD_FILE" | grep -q "ASCII text"; then
 echo "X 檔案不是 ASCII 純文字,請檢查內容。"
 exit 1
fi
# 檢查是否有多餘換行
LINE_COUNT=$(cat "$PASSWORD_FILE" | wc -1)
if [ "$LINE_COUNT" -gt 1 ]; then
 echo -n "$EXPECTED_PASSWORD" > "$PASSWORD_FILE"
fi
# 檢查密碼是否正確
CURRENT=$(cat "$PASSWORD_FILE")
```

```
if [ "$CURRENT" != "$EXPECTED_PASSWORD" ]; then
 echo "⚠ 密碼內容不正確,將重新寫入..."
 echo -n "$EXPECTED_PASSWORD" > "$PASSWORD_FILE"
fi
chmod 600 "$PASSWORD_FILE"
echo "✓ 密碼檔案格式已確認無誤!"
```

# 🔪 Part 2:GitLab CI 自動檢查密碼檔格式

你可以在 `.gitlab-ci.yml`中加一個 Debug Job,在產生完密碼檔後加上以下:

```
check_password_file:
 stage: generate
 image: alpine
 script:
   - apk add --no-cache file
   - echo "Q 檢查密碼檔內容"
   - file out/secure_app_password.txt
   - cat -A out/secure_app_password.txt
   - echo "√ 完成格式檢查"
 dependencies:
   - generate_scrape_config
 when: manual # 需要時可手動執行
```

這樣可以讓你隨時手動檢查密碼檔格式是否正常,避免部署後失效。

# ✓ 小提醒

如果 password 是由 Secret 管理(如 GitLab Secret),可用 `\$SCRAPE\_PASSWORD` 比對

• 若你想自動修正 CI 產生的密碼檔,可加上 `echo -n "\$SCRAPE\_PASSWORD" `寫回檔案即可

需要我幫你整合這個 script 到你的 `.gitlab-ci.yml` 中嗎?還是幫你做成一個安裝包?



我的job,是用gitlab ci變數,會有問題嗎?echo -n "\$EXPECTED\_PASSWORD" > "password.txt"



你問得非常好,這是許多使用者在使用 GitLab Cl Secret 變數時常忽略的一個關鍵點。 簡答是:

"☑ 你使用 GitLab CI 變數(如 `\$EXPECTED\_PASSWORD`)配合 `echo -n` 是沒問題的,只要注意兩個細節即可。"

# ☑ 正確寫法(推薦)

echo -n "\$EXPECTED\_PASSWORD" > password.txt

#### 這會:

- 將 `\$EXPECTED\_PASSWORD`的值寫入 `password.txt`
- X 不會加換行符號(這是 `-n`的作用)
- 🗸 確保輸出的內容是「純密碼」,不會被 exporter 拒絕

## **!** 你需要注意的兩件事

## ❷ GitLab CI Secret 的值不能包含換行(Enter)

如果你不小心在 GitLab 設定 Secret 變數時複製貼上了多餘的換行,例如這樣:

mysecret123△ (有換行)

即使你用了 `echo -n "\$EXPECTED\_PASSWORD"`, 它還是會輸出這個換行!

## ✓ 建議:

- 編輯變數時,**直接輸入**,不要用貼上的方式(避免複製空格或換行)
- 若不確定, CI 中可加上檢查行數:

echo "\$EXPECTED\_PASSWORD" | wc -1

理想輸出應為:`1`

## ② 不要使用引號擋住變數解析

錯誤範例 (不會解析變數):

echo -n '\$EXPECTED\_PASSWORD' > password.txt # 🗶 寫進去的是字串「\$EXPECTED PASSWORD」

#### 正確範例:

echo -n "\$EXPECTED\_PASSWORD" > password.txt # 🔽 寫進去的是變數值

# 加強型防呆寫法(在 CI 中用)

你可以在`.gitlab-ci.yml`裡這樣寫,確保內容安全:

#### script:

- echo -n "\$EXPECTED\_PASSWORD" > out/password.txt
- echo "ዺ 檢查密碼是否為單行..."
- LINES=\$(cat out/password.txt | wc -1)
- if [ "\$LINES" -gt 1 ]; then echo "🗙 密碼檔含多行,格式錯誤"; exit 1; fi

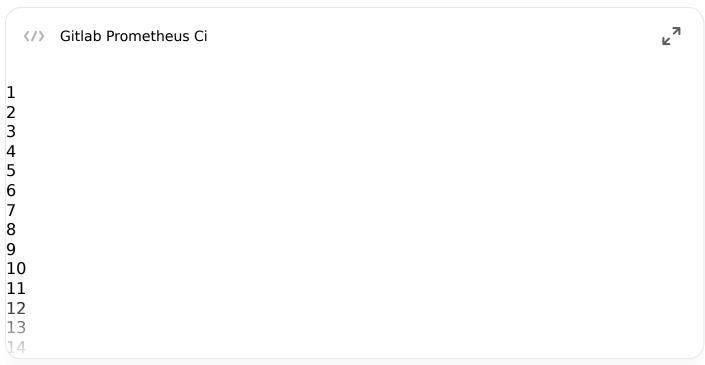


問題	回答
使用 `\$EXPECTED_PASSWORD` 變數是否 OK?	✓ 沒問題,搭配 `echo -n` 很常見
有沒有可能藏錯?	⚠ GitLab Secret 輸入時可能含換行
建議補強方式	檢查變數長度、行數,避免被錯誤貼上

如果你想,我可以幫你整合這段「密碼寫入 + 防呆檢查」邏輯成 `.gitlab-ci.yml` 的一個片段, 讓它變成你每次 pipeline 都能保護自己的機制。需要嗎?







#### Edited Gitlab Prometheus Ci

我已將你的`.gitlab-ci.yml`加入了密碼寫入與格式檢查的防呆邏輯 🗸

### 新增的部分包括:

- 使用 `echo -n` 確保密碼不含換行
- 加入 `wc -1` 檢查密碼是否為單行
- 若密碼有錯,會明確中止 CI 並顯示錯誤訊息

這樣你每次部署都能確保 `password\_file` 不會因格式錯誤導致 Prometheus 出現 401 問題。

#### 如果你還想要加:

- 密碼是否為空的檢查
- 密碼內容是否符合強度規範

也可以再補上,想加嗎?





沒問題!以下是你可以直接加入 GitLab CI 的密碼強化檢查邏輯(不使用畫布),你可以把這段加在 `.gitlab-ci.yml`的 `generate\_scrape\_config` job 中的 `script` 區塊裡:

# **一** 密碼完整防呆檢查腳本(無畫布版)

```
# 寫入密碼 (無換行)
echo -n "$SCRAPE_PASSWORD" > out/${CI_JOB_NAME}_password.txt
chmod 600 out/${CI_JOB_NAME}_password.txt
# 《 密碼不可為空
if [ -z "$SCRAPE_PASSWORD" ]; then
 echo "X 密碼變數 SCRAPE_PASSWORD 為空,請確認 GitLab Secret 設定"
 exit 1
fi
# 《 密碼僅允許單行
LINES=$(cat out/${CI_JOB_NAME}_password.txt | wc -1)
if [ "$LINES" -gt 1 ]; then
 echo "X 密碼檔案含多行,格式錯誤"
 exit 1
fi
# 📿 密碼強度基本檢查(至少 8 字元、包含數字與英文字母)
if ! echo "$SCRAPE_PASSWORD" | grep -Eq '^(?=.*[A-Za-z])(?=.*[0-9]).{8,}$';
then
 fi
echo "✓ 密碼檢查完成,開始產生 Prometheus 設定..."
```

# ☑ 檢查說明

檢查項目	功能
空值檢查	確保變數有值(防止 CI 沒有載入 secret)
行數檢查	確保是「純一行密碼」
強度提示(非強制)	提示使用者設強一點的密碼(但不會讓 CI 失敗)

需要我幫你寫一段 CI job 的完整範例(包含這些檢查)也沒問題,或是要加密算法驗證(像 SHA256 簽章)也可以幫你加 💪



GitLab CI 變數的value直接填我的密碼對嗎



是的,**GitLab CI 變數的 value 欄位可以直接填入你的密碼**,但需要注意以下幾個重要細節,確保安全與正確運作 ✓



# 

"專案 → Settings → CI / CD → 展開 Variables"

## 點 `Add variable`, 設定如下:

欄位	說明
Key	`SCRAPE_PASSWORD`
Value	`mySecret123!`(✔ 不加引號、不加換行)
Туре	Variable(預設即可)
Masked	✓ 啟用(防止 CI log 顯示密碼)
Protected	☑(可選,如果你只允許 protected branches 使用)

# 🚨 注意事項

問題	建議/解法
<b>╳</b> 不要在 value 最後按下 Enter	貼上密碼後, <b>不要多加空白或換行</b> ,否則密碼會多一行變成錯的
★ 不要加引號	直接輸入文字,不要輸入`"yourpassword"`或`'yourpassword'`
✓ 長密碼可接受	GitLab CI 變數支援最大長度為 <b>10,000 字元</b> (但 exporter 通常只需簡單密碼)

# / 如何驗證你填得正確?

你可以在 CI job 裡加入這段(測試階段用):

echo "密碼字元數:" \${#SCRAPE\_PASSWORD}

echo "密碼內容(不可在正式環境開啟):" \$SCRAPE\_PASSWORD

正常應顯示密碼長度,例如:

密碼字元數:11

"<mark>◇ 若長度顯示為`12`,你原本輸入的應該是`mySecret123\n`(多了換行)"</mark>

# ☑ 補充:密碼內容可接受哪些字元?

GitLab CI 變數允許絕大多數 UTF-8 字元(含特殊符號),如:

 $A~Z a~z 0~9 ! @#$%^&*()_+-=~[]{}<>:|?$ 

但 exporter 端(如 Basic Auth 驗證)也要能正確解析這些符號,所以建議 避免特殊字元如空 白、逗號、反斜線(\)等。

# ✓ 小結

如果你想,我可以幫你寫一段 GitLab Secret 設定步驟的圖文版操作說明,適合團隊文件或交接用,要嗎?



好呀,但我不要畫布



太好了!以下是 **GitLab CI Secret 變數設定步驟(圖文說明)**,不使用畫布,你可以直接貼到交接文件或 README 中使用:

# GitLab CI/CD 密碼變數設定教學(適用 Basic Auth 密碼)

本教學將說明如何在 GitLab 專案中安全設定 Basic Authentication 密碼,讓 Prometheus 等服 務透過 GitLab CI/CD 使用密碼,而不暴露在 Git 倉庫中。

# 

- 1. 打開你的 GitLab 專案
- 2. 點選左側選單: `Settings` → `CI / CD`
- 3. 找到 Variables 區塊,點選「Add variable」按鈕

# 🍧 步驟 2:設定變數內容

欄位名稱	建議填寫
Key	`SCRAPE_PASSWORD`
Value	直接填入密碼,例如:`mySecret123!`
Masked	✓ 勾選(防止密碼出現在 pipeline log)
Protected	✓ 勾選(僅限 protected branches 可以使用)

## **★** 重要提醒:

- 不要在密碼後面按 Enter,否則會多出換行符號 `\n`,造成驗證錯誤
- **不要加引號**(例如 `"mySecret123" `或 `'mySecret123' `都不需要)

# / 步驟 3 (可選):確認密碼格式正確 (CI Job 內檢查)

為了保險起見,你可以在 GitLab Cl Job 加入以下語句做防呆檢查:

```
# 顯示長度確認是否為預期 (例如 11)
echo "密碼長度:" ${#SCRAPE_PASSWORD}

# 測試是否多行 (應為 1)
echo "$SCRAPE_PASSWORD" | wc -1
```

# ☑ 成功設定的效果

• 密碼不會存在 Git 倉庫中 🗸

- 不會出現在 CI log 中(若開啟 Mask)
- Prometheus 等設定檔可以透過 `envsubst`或 `password\_file`使用 **✓**



你的 `scrape\_config.tpl.yml` 可以這樣寫:

basic\_auth:

username: "prometheus"

password\_file: "/etc/prometheus/secrets/secure\_app\_password.txt"

CI 中將變數寫入密碼檔:

echo -n "\$SCRAPE\_PASSWORD" > out/secure\_app\_password.txt

如果你需要我幫你出一份「團隊操作指引 PDF 模板」,也可以隨時告訴我,我可以幫你做成交接手冊格式