



# GitLab 白名單設置



gitlab.yml, 我需要能設置白名單控制job能不能執行, 例如白名單: white-1/white-2/apple, 代表只有containername=white-1或white-2或apple才可以執行, "white"沒有完全匹配就不能執行, 我的白名單很多, 所以給我通用的設置白名單方式



要在 GitLab CI/CD 的 `.gitlab-ci.yml` 文件中設置白名單來控制 Job 是否執行, 可以利用 CI/CD 變數和腳本的邏輯判斷來實現。以下是一個通用的設置範例：

```
stages:
  - test

variables:
  WHITELIST: "white-1,white-2,apple" # 定義白名單, 以逗號分隔

check_whitelist:
  stage: test
  script:
    - echo "Container Name: $CONTAINER_NAME"
    - |
      # 將白名單轉換為陣列, 檢查當前名稱是否在白名單中
      IFS=',' read -r -a WHITELIST_ARRAY <<< "$WHITELIST"
      ALLOWED=false
      for NAME in "${WHITELIST_ARRAY[@]}"; do
        if [[ "$CONTAINER_NAME" == "$NAME" ]]; then
          ALLOWED=true
          break
        fi
      done

      if [[ "$ALLOWED" == "true" ]]; then
        echo "Container name is in the whitelist. Proceeding with the job..."
      else
        echo "Container name is NOT in the whitelist. Exiting job."
        exit 1
```



```
fi
- echo "Job execution continues..."
only:
- branches # 可根據需要限制在哪些分支執行
```

## 說明：

### 1. ``WHITELIST``:

- 使用變數 ``WHITELIST`` 定義容器名稱白名單，並用逗號分隔。

### 2. ``CONTAINER_NAME``:

- 假設 ``CONTAINER_NAME`` 是你的容器名稱，這個變數可以在 GitLab CI/CD Pipeline 中設置（例如手動指定或從上下文獲取）。

### 3. 白名單檢查邏輯:

- 使用 Bash 腳本將白名單轉換為陣列，逐一檢查容器名稱是否完全匹配。
- 如果匹配，則設置 ``ALLOWED=true``，否則退出 Job。

### 4. ``only``:

- 限制 Job 執行的條件，例如只在特定分支上執行。

## 使用方式：

- 在觸發 Pipeline 時，確保通過 CI/CD 變數（例如 ``CONTAINER_NAME``）提供容器名稱。
- 例：觸發 Pipeline 並設置 ``CONTAINER_NAME`` 為 ``white-1``：

```
curl --request POST \
  --form "token=<your-trigger-token>" \
  --form "ref=main" \
  --form "variables[CONTAINER_NAME]=white-1" \
  https://gitlab.example.com/api/v4/projects/<your-project-id>/trigger/pipeline
```

## 注意事項：



- 白名單很長時，只需要在 `WHITELIST` 變數中更新即可，腳本邏輯不需要修改。
- 若需要更動白名單，建議透過 CI/CD Variables 管理，避免硬編碼。



WHITELIST: "white-1/white-2/apple"可以嗎



當然可以將白名單格式改成用斜線（`/`）分隔，例如 `WHITELIST: "white-1/white-2/apple"`，只需要在腳本中稍微調整分隔符即可。以下是修改後的 `.gitlab-ci.yml`：

```
stages:
  - test

variables:
  WHITELIST: "white-1/white-2/apple" # 白名單，用斜線分隔

check_whitelist:
  stage: test
  script:
    - echo "Container Name: $CONTAINER_NAME"
    - |
      # 將白名單以斜線分隔並轉換為陣列
      IFS='/' read -r -a WHITELIST_ARRAY <<< "$WHITELIST"
      ALLOWED=false
      for NAME in "${WHITELIST_ARRAY[@]}"; do
        if [[ "$CONTAINER_NAME" == "$NAME" ]]; then
          ALLOWED=true
          break
        fi
      done

      if [[ "$ALLOWED" == "true" ]]; then
        echo "Container name is in the whitelist. Proceeding with the job..."
      else
        echo "Container name is NOT in the whitelist. Exiting job."
        exit 1
      fi
```



```
- echo "Job execution continues..."
only:
- branches # 根據需要限制執行範圍
```

## 說明更新：

### 1. 白名單格式：

- 現在白名單用 `/` 分隔，例如 `white-1/white-2/apple`。

### 2. 修改分隔符：

- 用 `IFS='/ '` 來分隔白名單內容，將其轉換為陣列。

### 3. 其他邏輯不變：

- 剩下的邏輯與逗號分隔方式一致，逐一檢查容器名稱是否在白名單中。

## 使用方式：

與之前的方式一致，只需要設置 `CONTAINER\_NAME` 並確保 `WHITELIST` 符合新格式。

## 範例測試：

當 `WHITELIST` 設置為 `"white-1/white-2/apple"`，並且 `CONTAINER\_NAME` 分別設置為以下值：

- `white-1`：允許執行。
- `apple`：允許執行。
- `white`：不允許執行（不完全匹配）。
- `unknown`：不允許執行。



如果要改成用job的rules:來控制執行條件，而不是script可以嗎





當然可以！如果你想使用 `rules:` 來控制 Job 的執行條件，而不是在 `script` 裡用腳本判斷，可以利用 GitLab CI/CD 的變數條件來實現。以下是一個範例，使用白名單 `WHITELIST` 和 `rules:` 控制 Job 是否執行：

```
stages:
  - test

variables:
  WHITELIST: "white-1/white-2/apple" # 白名單，用斜線分隔

check_whitelist:
  stage: test
  rules:
    - if: '$CONTAINER_NAME =~ /^(white-1|white-2|apple)$/' # 完全匹配白名單中的值
      when: on_success
    - when: never # 其他情況不執行
  script:
    - echo "Job is running for container: $CONTAINER_NAME"
```

## 說明

### 1. 使用 `rules:` 判斷條件：

- `if: '$CONTAINER_NAME =~ /^(white-1|white-2|apple)$/'` 表示只執行當 `$CONTAINER_NAME` 的值匹配正則表達式時。
- 正則表達式 `^(white-1|white-2|apple)$` 表示完全匹配 `white-1`、`white-2` 或 `apple`。

### 2. 執行與不執行的條件：

- 當 `$CONTAINER_NAME` 符合白名單條件時執行 Job (`when: on_success`)。
- 當不符合條件時，直接設定 `when: never`，Job 完全不執行。

### 3. 變數白名單的靈活性：

- 如果白名單比較長，可以通過 GitLab CI/CD Variables 管理（比如環境變數 `WHITELIST`），在 `rules:` 的條件裡設置正則表達式動態生成（需要更進階的腳本支持）。



## 更新白名單

若要更新白名單，只需要修改 `rules:` 裡的正則表達式，例如：

```
if: '$CONTAINER_NAME =~ /^(name1|name2|name3|more-names)$/'
```

或直接將白名單值拆成變數（若支持複雜處理）。

## 優點

- **更簡潔**：直接用 `rules:` 判斷，不需要寫額外的腳本。
- **執行效率高**：不符合條件時，Job 完全不執行，避免不必要的資源消耗。

## 適用場景

- 當你的白名單規則固定，並且可以通過正則表達式描述時，這種方式是最簡潔的解法。



white-1|white-2|app，可換成WHITELIST變數嗎



直接在 `rules:` 中使用變數如 `WHITELIST` 並進行正則判斷是比較受限的，因為 GitLab 的 `rules:` 不支持直接解析變數來構建動態的正則表達式。但可以使用一些變通的方法。

以下是一個接近你需求的設計，利用 `rules:` 和環境變數來控制執行：

## 方案一：事先展開正則表達式

如果白名單可以提前展開成正則格式，則在 `.gitlab-ci.yml` 中直接使用變數拼接正則表達式：

```
stages:
  - test

variables:
```





```
WHITELIST: "^(white-1|white-2|apple)$" # 白名單的正則形式

check_whitelist:
  stage: test
  rules:
    - if: '$CONTAINER_NAME =~ $WHITELIST'
      when: on_success
    - when: never
  script:
    - echo "Job is running for container: $CONTAINER_NAME"
```

說明：

- `WHITELIST` 定義成正則表達式的形式，例如 `^(white-1|white-2|apple)\$`。
- `rules:` 中的條件 `\$CONTAINER\_NAME =~ \$WHITELIST` 直接使用此變數進行正則匹配。

## 方案二：通過外部腳本生成正則

如果白名單內容很多或頻繁變動，可以通過腳本將白名單動態生成正則表達式。示例：

```
stages:
  - test

variables:
  WHITELIST: "white-1/white-2/apple" # 白名單，用斜線分隔

check_whitelist:
  stage: test
  rules:
    - if: '$CONTAINER_NAME =~ $WHITELIST_REGEX'
      when: on_success
    - when: never
  before_script:
    - export WHITELIST_REGEX="^($(echo $WHITELIST | tr '/' '|'))$"
  script:
    - echo "Job is running for container: $CONTAINER_NAME"
```

說明：



### 1. 白名單格式：

- ``WHITELIST`` 使用 ``/`` 分隔，例如 ``white-1/white-2/apple``。

### 2. 正則生成：

- 在 ``before_script`` 中將 ``WHITELIST`` 轉換為正則表達式，使用 ``tr '/' '|'`` 替換 ``/`` 為 ``|``，生成 ``^(white-1|white-2|apple)$``。
- 然後將結果賦值給 ``WHITELIST_REGEX``。

### 3. 匹配邏輯：

- 在 ``rules:`` 中使用動態生成的 ``WHITELIST_REGEX``。

## 方案比較

特性	方案一	方案二
白名單更新成本	必須手動修改 YAML	通過變數即可自動處理
白名單長度支持	適合固定、簡短在白名單	適合白名單較長或頻繁變動的情況
實現簡單性	最簡單的方式	需要額外處理，但更靈活

根據需求選擇更適合的方案。如果白名單頻繁變動，建議使用 **方案二**，結合動態正則生成方式會更靈活。



改成 `WHITELIST:"white-1/white-2/apple"`，  
`CONTAINER_NAME:white`、`hite` 可以執行，`1hite` 不能執行



如果白名單為 ``WHITELIST: "white-1/white-2/apple"``，且希望以下行為：

### 1. 完全匹配或部分匹配：

- ``CONTAINER_NAME=white`` 可以執行。
- ``CONTAINER_NAME=hite`` 可以執行。
- ``CONTAINER_NAME=1hite`` **不能執行**（因為不包含 ``white`` 或 ``hite`` 作為子串）。





這可以通過調整 `rules:` 的條件來達成。在這種情況下，可以使用動態生成的正則表達式來實現部分匹配：

## 解決方案

```
stages:
  - test

variables:
  WHITELIST: "white-1/white-2/apple" # 白名單，用斜線分隔

check_whitelist:
  stage: test
  rules:
    - if: '$CONTAINER_NAME =~ $WHITELIST_REGEX'
      when: on_success
    - when: never
  before_script:
    # 將 WHITELIST 轉換為正則表達式，允許部分匹配
    - export WHITELIST_REGEX="$(echo $WHITELIST | tr '/' '|')'"
  script:
    - echo "Job is running for container: $CONTAINER_NAME"
```

## 說明

### 1. `WHITELIST`:

- 定義白名單為 `white-1/white-2/apple`，用斜線（`/`）分隔。

### 2. 動態生成正則表達式:

- 在 `before_script` 中，將白名單中的條目用 `|` 分隔，形成正則表達式。
- 生成的 `WHITELIST_REGEX` 為 `white-1|white-2|apple`。

### 3. 部分匹配規則:

- 使用 `rules:` 判斷 `CONTAINER_NAME` 是否包含白名單中的任意部分字串。
- 正則表達式自動支持子串匹配，因此 `CONTAINER_NAME=white` 和 `hite` 可以匹配，但 `1hite` 不行。



#### 4. 執行條件:

- 如果匹配白名單中的任意項目，則執行 Job。
- 否則，Job 不執行（``when: never``）。

### 範例行為

<code>`CONTAINER_NAME`</code>	結果	說明
<code>`white`</code>	執行	子串 <code>`white`</code> 匹配成功
<code>`hite`</code>	執行	子串 <code>`hite`</code> 匹配成功（部分匹配）
<code>`1hite`</code>	不執行	沒有與白名單完全匹配或部分匹配
<code>`apple`</code>	執行	完全匹配 <code>`apple`</code>
<code>`apples`</code>	不執行	子串部分匹配不適用於 <code>`apples`</code>

### 進階擴展

如果需要更複雜的匹配邏輯（例如忽略大小寫匹配等），可以進一步調整正則表達式的生成方式。例如加入 ``(?i)`` 前綴來啟用大小寫不敏感匹配。



我的check\_whitelist是由qqq這個job trigger的，CONTAINER\_NAME是qqq裡宣告的，我為什麼不能在check\_whitelist做export WHITELIST\_REGEX="(\$(echo \$WHITELIST | tr '/' '|'))"，rules:  
- if: '\$CONTAINER\_NAME =~ \$WHITELIST\_REGEX'



在 GitLab CI/CD 中，``rules:`` 的條件是解析靜態上下文的，它們只會在 Pipeline 創建階段（Pre-Execution Stage）解析，因此不能直接依賴 ``before_script`` 中的執行結果。例如，你在 ``check_whitelist`` 中的 ``export WHITELIST_REGEX`` 是執行時產生的變數，而 ``rules:`` 在 Job 執行之前就已經完成判斷，無法動態解析。





這是為什麼你不能在 ``rules:`` 使用 ``before_script`` 中生成的變數（如 ``$WHITELIST_REGEX``）的原因。

## 解決方法

### 方法 1：直接使用靜態正則表達式

如果白名單是靜態的（如 ``WHITELIST: "white-1/white-2/apple"``），你可以直接在 ``.gitlab-ci.yml`` 中寫入靜態正則表達式，而不依賴 ``before_script``：

```
variables:
  WHITELIST: "white-1/white-2/apple"

check_whitelist:
  stage: test
  rules:
    - if: '$CONTAINER_NAME =~ /(white-1|white-2|apple)/'
      when: on_success
    - when: never
  script:
    - echo "Job is running for container: $CONTAINER_NAME"
```

這種方式省略了 ``before_script`` 的動態生成，直接在 ``rules:`` 中寫出正則表達式。

### 方法 2：在觸發器 Job (``qqq``) 中生成正則表達式

你可以讓觸發 Job (``qqq``) 負責生成正則表達式並作為變數傳遞給被觸發的 Job (``check_whitelist``)。示例如下：

#### 1. ``qqq`` Job：

```
stages:
  - trigger

qqq:
```



```
stage: trigger
variables:
  WHITELIST: "white-1/white-2/apple"
script:
  # 將 WHITELIST 轉換為正則表達式並傳遞給下游 Job
  - export WHITELIST_REGEX="$(echo $WHITELIST | tr '/' '|'))"
  - echo "Triggering pipeline with WHITELIST_REGEX=$WHITELIST_REGEX"
  - curl --request POST \
    --form "token=<your-trigger-token>" \
    --form "ref=main" \
    --form "variables[CONTAINER_NAME]=white" \
    --form "variables[WHITELIST_REGEX]=$WHITELIST_REGEX" \
    https://gitlab.example.com/api/v4/projects/<your-project-id>/trigger/pipeline
```

## 2. `check\_whitelist` Job :

```
check_whitelist:
  stage: test
  rules:
    - if: '$CONTAINER_NAME =~ $WHITELIST_REGEX'
      when: on_success
    - when: never
  script:
    - echo "Job is running for container: $CONTAINER_NAME"
```

### 關鍵步驟：

#### 1. 在 `qqq` 中生成正則表達式：

- 利用 Shell 腳本生成正則表達式，並將其作為變數傳遞到下游 Pipeline。

#### 2. 使用觸發器傳遞變數：

- 利用 GitLab 的 [Pipeline Triggers](#) 傳遞生成的 `WHITELIST\_REGEX` 和 `CONTAINER\_NAME`。

#### 3. 在 `check\_whitelist` 中直接使用 `WHITELIST\_REGEX`：

- `rules:` 的條件能夠解析 `WHITELIST\_REGEX`，因為它在 Pipeline 創建時已靜態設置。



### 方法 3：用父子 Pipeline 傳遞變數

如果你不想用觸發器，可以改用父子 Pipeline，通過 `trigger` 關鍵字傳遞變數：

#### 1. 父 Pipeline (`qqq`)：

```
stages:
  - trigger

qqq:
  stage: trigger
  variables:
    WHITELIST: "white-1/white-2/apple"
  script:
    - export WHITELIST_REGEX="$(echo $WHITELIST | tr '/' '|')"
```

trigger:

- include:
  - local: child-pipeline.yml
- strategy: depend
- forward:
  - pipeline\_variables: true

#### 2. 子 Pipeline (`child-pipeline.yml`)：

```
stages:
  - test

check_whitelist:
  stage: test
  rules:
    - if: '$CONTAINER_NAME =~ $WHITELIST_REGEX'
      when: on_success
    - when: never
  script:
    - echo "Job is running for container: $CONTAINER_NAME"
```

這樣，父 Pipeline 的變數（如 `WHITELIST_REGEX` 和 `CONTAINER_NAME`）會自動傳遞到子 Pipeline，`rules:` 可以正確解析。



## 關鍵點

- ``rules:`` 的條件在 **Pipeline 創建階段解析**，只能使用當時可用的靜態變數。
- 動態生成的值（例如 ``before_script`` 中的結果）需要通過觸發器或父子 Pipeline 傳遞為靜態變數。
- 選擇適合你的工作流的方式（靜態正則、觸發器、或父子 Pipeline）來解決這個限制。



```
rules:  
  - if: '$WHITELIST_REGEX =~ $CONTAINER_NAME'才對吧
```



在 GitLab CI/CD 中，``rules:`` 的條件是基於靜態變數進行解析的。你希望用 ``rules:`` 條件表達的是：

```
rules:  
  - if: '$WHITELIST_REGEX =~ $CONTAINER_NAME'
```

但這樣的條件不會正確工作，因為 **GitLab 不支持在 ``rules:`` 中進行兩個變數之間的正則匹配**。

在 GitLab 中：

- ``rules: if:`` 的條件支持簡單的運算，如變數的字符串比較（``==``）、不等（``!=``），以及對靜態正則表達式的匹配。
- 但是，它不支持類似 ``$WHITELIST_REGEX =~ $CONTAINER_NAME`` 的雙變數匹配，因為 ``$WHITELIST_REGEX`` 不能作為動態正則表達式。

## 解決方案：轉變匹配邏輯





如果需要實現動態的正則匹配，請選擇以下解決方案之一。

### 方案 1：將白名單直接寫成正則表達式（避免使用動態正則）

直接使用靜態正則匹配白名單（如 `^(white-1|white-2|apple)$``），將正則固定為靜態字符串，並在 `rules:`` 中判斷：

```
stages:
  - test

variables:
  WHITELIST_REGEX: "^(white-1|white-2|apple)$" # 靜態正則表達式

check_whitelist:
  stage: test
  rules:
    - if: '$CONTAINER_NAME =~ $WHITELIST_REGEX'
      when: on_success
    - when: never
  script:
    - echo "Job is running for container: $CONTAINER_NAME"
```

#### 特點：

- 不依賴動態生成的正則。
- 靜態的 `WHITELIST_REGEX`` 在 `rules:`` 條件中可直接使用。

### 方案 2：使用觸發器生成正則並傳遞

讓觸發器（或父 Pipeline）負責生成靜態正則並將其傳遞給子 Pipeline，實現動態傳遞。

#### 觸發器 Job 示例：

```
qqq:
  stage: trigger
```



```
variables:
  WHITELIST: "white-1/white-2/apple"
script:
  - export WHITELIST_REGEX="^(white-1|white-2|apple)$"
  - echo "Triggering child job with WHITELIST_REGEX=$WHITELIST_REGEX"
  - curl --request POST \
    --form "token=<your-trigger-token>" \
    --form "ref=main" \
    --form "variables[CONTAINER_NAME]=white" \
    --form "variables[WHITELIST_REGEX]=$WHITELIST_REGEX" \
    https://gitlab.example.com/api/v4/projects/<your-project-id>/trigger/pipeline
```

### 被觸發 Job 示例：

```
check_whitelist:
  stage: test
  rules:
    - if: '$CONTAINER_NAME =~ $WHITELIST_REGEX'
      when: on_success
    - when: never
  script:
    - echo "Job is running for container: $CONTAINER_NAME"
```

---

### 方案 3：用 `script` 實現判斷（非 `rules`）

如果非要用動態正則判斷，但不想在父 Job 做處理，可以將判斷邏輯放在 `script` 中，而 `rules` 僅控制基礎條件。

### 配置：

```
stages:
  - test

variables:
  WHITELIST: "white-1/white-2/apple"

check_whitelist:
```





```
stage: test
script:
  - export WHITELIST_REGEX="^($(echo $WHITELIST | tr '/' '|'))$"
  - if [[ "$CONTAINER_NAME" =~ $WHITELIST_REGEX ]]; then
    echo "Job is running for container: $CONTAINER_NAME";
  else
    echo "Container $CONTAINER_NAME is not allowed";
    exit 1;
  fi
```

### 特點：

- 動態生成的正則可以在執行時解析。
- 不依賴 GitLab 的 `rules:` 條件邏輯。

### 為什麼 `rules:` 無法解析 `\$(echo \$WHITELIST\_REGEX =~ \$CONTAINER\_NAME)`？

GitLab 的 `rules:` 是在 Pipeline 創建時解析的，而 `\$(echo \$WHITELIST\_REGEX =~ \$CONTAINER\_NAME)` 是執行時生成的，並不能用於 `rules:` 條件。因此，需要通過靜態正則或者提前生成的正則來實現。



方案 1 符合嗎，當 WHITELIST 設置為 "white-1/white-2/apple"，並且 CONTAINER\_NAME 分別設置為以下值：

white-1：允許執行。  
apple：允許執行。  
white：不允許執行（不完全匹配）。  
unknown：不允許執行。



方案 1 可以實現這種行為，只需稍微調整白名單的正則表達式生成方式，確保精確匹配白名單中的項目，而不是部分匹配。以下是具體配置：

### 正確的 `.gitlab-ci.yml` 配置



```
stages:
  - test

variables:
  WHITELIST: "white-1/white-2/apple" # 白名單，用斜線分隔
  WHITELIST_REGEX: "^\((white-1)|(white-2)|(apple))$" # 精確匹配的正則表達式

check_whitelist:
  stage: test
  rules:
    - if: '$CONTAINER_NAME =~ $WHITELIST_REGEX'
      when: on_success
    - when: never
  script:
    - echo "Job is running for container: $CONTAINER_NAME"
```

---

## 配置解釋

### 1. `WHITELIST\_REGEX` :

- 使用正則表達式 `^\((white-1)|(white-2)|(apple))\$`，確保只允許完全匹配 `white-1`、`white-2` 或 `apple`。
- `^` 和 `\$` 用於確保整個字符串匹配，而不是部分匹配。

### 2. `rules:` 條件：

- `\$CONTAINER\_NAME =~ \$WHITELIST\_REGEX`：如果 `CONTAINER\_NAME` 與正則表達式匹配，則允許執行。
- 否則，Job 不執行（`when: never`）。

---

## 測試行為

當配置如下時：



CONTAINER_NAME	是否允許執行	說明
`white-1`	✓	完全匹配 `white-1`
`white-2`	✓	完全匹配 `white-2`
`apple`	✓	完全匹配 `apple`
`white`	✗	不允許部分匹配
`unknown`	✗	不在白名單中

## 如果需要動態生成 `WHITELIST\_REGEX`

如果白名單較長，可以通過外部腳本或觸發器生成正則表達式，並傳遞到 CI/CD 中。以下是如何動態生成的示例：

```
stages:
  - test

variables:
  WHITELIST: "white-1/white-2/apple" # 白名單，用斜線分隔

check_whitelist:
  stage: test
  before_script:
    - export WHITELIST_REGEX="^($(echo $WHITELIST | tr '/' '|'))$"
  rules:
    - if: '$CONTAINER_NAME =~ $WHITELIST_REGEX'
      when: on_success
    - when: never
  script:
    - echo "Job is running for container: $CONTAINER_NAME"
```

這樣，即使白名單更新，也不需要手動調整正則表達式，只需更新 `WHITELIST` 即可。