



# GitLab 白名單控制



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project\_namespace=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



在 GitLab CI/CD 中, 可以通过使用 `rules` 和环境变量的组合来实现白名单控制特定 Job 是否执行。以下是一个通用的 `.gitlab-ci.yml` 配置示例, 可以根据项目的 `CI_PROJECT_NAMESPACE` 来控制 Job 是否执行:

## 通用配置 `.gitlab-ci.yml`

```
stages:
  - test

# 定义一个白名单变量
variables:
  WHITELIST: "white-1,white-2,apple"

job_with_whitelist:
  stage: test
  script:
    - echo "This job runs only for whitelisted namespaces."
  rules:
    # 使用脚本检查是否属于白名单
    - if: '$CI_PROJECT_NAMESPACE && $WHITELIST =~ (?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)'
      when: on_success
    - when: never
```

## 关键部分说明

### 1. `WHITELIST` 变量:

- 定义一个包含多个白名单值的逗号分隔字符串, 例如 `white-1,white-2,apple`。
- 可以根据需要扩展或修改白名单内容。



## 2. `rules` 关键字：

- 使用 `if` 条件检查 `CI\_PROJECT\_NAMESPACE` 是否在 `WHITELIST` 中。
- 正则表达式 `(?:^|,)\$CI\_PROJECT\_NAMESPACE(?:,|\$)` 用于精确匹配单独的名称，避免部分匹配问题。

## 3. 条件规则：

- 如果 `CI\_PROJECT\_NAMESPACE` 属于白名单，Job 会正常执行。
- 如果不在白名单中，Job 被设置为 `when: never`，不会执行。

## 白名单扩展

如果白名单非常长，可以通过环境变量管理，例如使用 CI/CD Variables 在 GitLab Web 界面中配置全局的 `WHITELIST`，避免直接在 `.gitlab-ci.yml` 中硬编码。

- 前往 GitLab 项目的 `Settings > CI/CD > Variables`。
- 添加 `WHITELIST` 环境变量，值为 `white-1,white-2,apple,...`。

## 测试方法

1. 提交代码到测试项目。
2. 确保 `CI\_PROJECT\_NAMESPACE` 设置正确。
3. 检查是否正确触发符合白名单规则的 Job。

这样可以灵活地通过白名单控制 Job 的执行，且易于维护和扩展。



請詳細說明检查是否属于白名单



检查是否属于白名单的核心逻辑是使用条件语句和正则表达式来验证当前的 `CI\_PROJECT\_NAMESPACE` 是否出现在白名单中。这部分主要在 `rules` 的 `if` 条件中完成。

## 逻辑细节



## 1. 白名单存储:

- 将白名单值存储为一个逗号分隔的字符串，例如：``WHITELIST="white-1,white-2,apple"``。
- 使用 GitLab 的环境变量 (``variables``) 管理白名单，便于后续扩展。

## 2. 验证 ``CI_PROJECT_NAMESPACE``:

- 使用 GitLab 提供的环境变量 ``$CI_PROJECT_NAMESPACE``，它表示当前项目的命名空间（通常是项目的 Group）。
- 将 ``$CI_PROJECT_NAMESPACE`` 与 ``WHITELIST`` 中的值进行逐一比对，确保精确匹配（避免部分匹配）。

## 3. 正则表达式匹配:

- 正则表达式 ``(?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)`` 用于精确匹配，解析如下：
  - ``(?:^|,)``: 匹配开头或逗号，确保命名空间是从开头或前一个值的末尾开始。
  - ``$CI_PROJECT_NAMESPACE``: 当前项目命名空间。
  - ``(?:,|$)``: 匹配逗号或字符串结尾，确保命名空间是最后一个值或后面有分隔符。
- 这种方式避免了部分匹配问题，例如：
  - 如果白名单中有 ``white-1``，而项目命名空间是 ``white-10``，正则表达式会阻止错误匹配。

## 4. 条件规则:

- 通过 ``rules`` 的 ``if`` 条件检查：

```
- if: '$CI_PROJECT_NAMESPACE && $WHITELIST =~ (?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)'
```

- ``$WHITELIST =~ ...``: 使用正则表达式匹配 ``CI_PROJECT_NAMESPACE``。
- ``when: on_success``: 如果匹配成功，Job 执行。
- ``when: never``: 如果不匹配，Job 不执行。

## 示例代码解读



```
job_with_whitelist:
  stage: test
  script:
    - echo "This job runs only for whitelisted namespaces."
  rules:
    - if: '$CI_PROJECT_NAMESPACE && $WHITELIST =~ (?:^|,)$CI_PROJECT_NAMESPACE(?:,|$) '
      when: on_success
    - when: never
```

## 运行逻辑：

### 1. 检查变量是否存在:

- ``$CI_PROJECT_NAMESPACE``：确保项目命名空间变量存在。
- ``$WHITELIST``：确保白名单变量存在。

### 2. 正则表达式匹配:

- 示例：
  - ``WHITELIST="white-1,white-2,apple"``.
  - ``CI_PROJECT_NAMESPACE="white-1"``:
    - 正则匹配：``(?:^|,)white-1(?:,|$)``。
    - 匹配成功，Job 执行。
  - ``CI_PROJECT_NAMESPACE="white"``:
    - 正则匹配失败，Job 不执行。

### 3. 控制 Job 执行:

- 如果匹配成功，``when: on_success`` 执行 Job。
- 如果匹配失败，``when: never`` 跳过 Job。

## 测试匹配规则

为了确认真则表达式是否正常工作，可以使用简单的 Shell 脚本测试：

### 测试脚本

```
WHITELIST="white-1,white-2,apple"
CI_PROJECT_NAMESPACE="white-1"
```



```
if [[ $WHITELIST =~ (?:^|,)$CI_PROJECT_NAMESPACE(?:,|$) ]]; then
    echo "Namespace is in the whitelist."
else
    echo "Namespace is NOT in the whitelist."
fi
```

## 测试结果

1. ``CI_PROJECT_NAMESPACE="white-1"``:
  - 输出：``Namespace is in the whitelist.``
2. ``CI_PROJECT_NAMESPACE="white"``:
  - 输出：``Namespace is NOT in the whitelist.``

## 总结

这种方法使用环境变量存储白名单，通过正则表达式精确匹配 ``CI_PROJECT_NAMESPACE``，实现灵活、可维护的 Job 白名单控制。



`(?:^|,)$CI_PROJECT_NAMESPACE(?:,|$) ??`



``(?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)`` 是一个正则表达式，用于精确匹配白名单中的项目命名空间 (``CI_PROJECT_NAMESPACE``)，确保匹配时不会误匹配到部分字符串。下面详细解释这个正则表达式的每一部分及其作用。

## 正则表达式结构拆解

```
(?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)
```

1. ``(?:^|,)``
  - 作用：匹配字符串的开头 ``^`` 或逗号 ``,``。
  - 解析：



- ``^``：匹配字符串的开始位置。
- ```,``：匹配白名单中逗号分隔的值。
- ``(?:...)``：非捕获组，表示将这一部分作为逻辑分组但不捕获到匹配结果中。
- 意义：确保 ``SCI_PROJECT_NAMESPACE`` 是白名单的第一个值或紧跟在一个逗号之后。

## 2. ``SCI_PROJECT_NAMESPACE``

- 作用：动态插入项目的命名空间（例如 ``white-1``、``apple``）。
- 意义：这个值将从 GitLab 的环境变量中获取，表示当前运行的项目命名空间。

## 3. ``(?:,|$)``

- 作用：匹配字符串的结尾 ``$`` 或逗号 ```,``。
- 解析：
  - ```,``：匹配白名单值之后的逗号。
  - ``$``：匹配字符串的结尾。
  - ``(?:...)``：非捕获组，和上面一样，用于逻辑分组而不捕获。
- 意义：确保 ``SCI_PROJECT_NAMESPACE`` 是白名单的最后一个值或其后紧跟逗号分隔的值。

---

## 正则表达式整体匹配逻辑

- 匹配的字符串模式为：
  - ``SCI_PROJECT_NAMESPACE`` 是白名单中的完整值，且：
    - 是白名单的第一个值。
    - 或者是白名单中以逗号分隔的值。
    - 或者是白名单的最后一个值。

---

## 举例分析

假设 ``WHITELIST="white-1,white-2,apple"``



### 情况 1：`CI\_PROJECT\_NAMESPACE="white-1"`

- 字符串：`"white-1,white-2,apple"`
  - 匹配流程：
    - `(?:^|,)`：匹配开头（`^`）。
    - `white-1`：匹配当前项目命名空间。
    - `(?:,|\$)`：匹配后续的逗号（`,`）。
  - 结果：匹配成功。
- 

### 情况 2：`CI\_PROJECT\_NAMESPACE="white"`

- 字符串：`"white-1,white-2,apple"`
  - 匹配流程：
    - `(?:^|,)`：匹配开头（`^`）。
    - `white`：不完整匹配，失败。
  - 结果：匹配失败。
- 

### 情况 3：`CI\_PROJECT\_NAMESPACE="apple"`

- 字符串：`"white-1,white-2,apple"`
  - 匹配流程：
    - `(?:^|,)`：匹配逗号（`,`）。
    - `apple`：匹配当前项目命名空间。
    - `(?:,|\$)`：匹配字符串结尾（`\$`）。
  - 结果：匹配成功。
- 

## 部分匹配问题的解决

使用这个正则表达式的好处是，它避免了误匹配，例如：



错误匹配示例（如果不使用 ``(?:^|,)`` 和 ``(?:,|$)``）：

- 白名单：``white-1,white-2,apple``
- 项目命名空间：``white``
- 如果仅使用 ``WHITELIST =~ $CI_PROJECT_NAMESPACE``，正则可能会错误匹配到 ``white-1`` 的前半部分 ``white``。

通过 ``(?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)``，确保只有完整值匹配时才会触发 Job。

## 总结

正则表达式 ``(?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)`` 的主要特点是：

1. 确保精确匹配白名单中的项目命名空间。
2. 避免部分匹配的问题。
3. 支持白名单的多值逗号分隔格式。
4. 灵活适用于 GitLab CI/CD 的动态环境变量。

这样就能有效地控制哪些命名空间的项目可以触发 Job。



dockercompose的部屬pipeline可用上述方法嗎



在使用 Docker Compose 的部署 Pipeline 中，也可以用上述方法结合 GitLab CI/CD 的 ``rules`` 和环境变量来控制特定命名空间是否可以运行 Pipeline。以下是详细实现方式和注意事项：

## 目标





控制 Pipeline 的执行，使得只有在特定 `CI\_PROJECT\_NAMESPACE`（白名单）下，Docker Compose 的部署 Job 才能执行。

## 示例 `.gitlab-ci.yml` 配置

```
stages:
  - build
  - deploy

variables:
  WHITELIST: "namespace-1,namespace-2,namespace-3"

build_job:
  stage: build
  script:
    - echo "Building Docker images..."
    - docker-compose build

deploy_job:
  stage: deploy
  script:
    - echo "Deploying using Docker Compose..."
    - docker-compose up -d
  rules:
    - if: '$CI_PROJECT_NAMESPACE && $WHITELIST =~ (?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)'
      when: on_success
    - when: never
```

## 配置说明

### 1. `variables` 部分

- 定义 `WHITELIST` 变量，存储允许的命名空间列表（逗号分隔）。
- 通过环境变量灵活配置白名单内容，便于后续维护。

### 2. `build\_job`

- 不受白名单限制，负责构建 Docker 镜像。



- 使用 ``docker-compose build`` 构建所需服务的镜像。

### 3. ``deploy_job``

- 部署时，检查 ``CI_PROJECT_NAMESPACE`` 是否在 ``WHITELIST`` 中：
  - ``$WHITELIST =~ (?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)``：精确匹配。
  - 如果匹配成功，部署 Job 执行。
  - 如果匹配失败，跳过部署 Job。

### 4. ``rules``

- 控制 Job 的执行条件：
    - 如果条件满足（命名空间在白名单中），Job 设置为 ``when: on_success``，正常执行。
    - 如果条件不满足，Job 设置为 ``when: never``，跳过执行。
- 

## 测试示例

### 情况 1：命名空间在白名单中

- 配置：
  - ``CI_PROJECT_NAMESPACE="namespace-1"``
  - ``WHITELIST="namespace-1, namespace-2, namespace-3"``
- 执行流程：
  - ``build_job`` 正常执行。
  - ``deploy_job`` 匹配白名单，执行成功。

### 情况 2：命名空间不在白名单中

- 配置：
  - ``CI_PROJECT_NAMESPACE="other-namespace"``
  - ``WHITELIST="namespace-1, namespace-2, namespace-3"``
- 执行流程：
  - ``build_job`` 正常执行。



- ``deploy_job`` 不匹配白名单，跳过执行。

### 情况 3：白名单为空

- 配置：
    - ``CI_PROJECT_NAMESPACE="namespace-1"``
    - ``WHITELIST=""``（未配置或为空）。
  - 执行流程：
    - ``build_job`` 正常执行。
    - ``deploy_job`` 规则不匹配，跳过执行。
- 

## 在 Docker Compose 部署中的优势

### 1. 白名单管理：

- 可以灵活地调整白名单，避免在不允许的命名空间中执行部署操作。

### 2. 避免误操作：

- 防止误将部署 Pipeline 执行到不应该的环境中（例如测试环境或非生产命名空间）。

### 3. 动态扩展：

- 如果白名单需要动态更新，可以通过 GitLab CI/CD 的 ``Settings > CI/CD > Variables`` 页面配置 ``WHITELIST``，无须更改 ``.gitlab-ci.yml`` 文件。
- 

## 注意事项

### 1. GitLab Runner 权限：

- 确保 GitLab Runner 有足够的权限运行 ``docker-compose`` 命令。
- 如果使用 Docker-in-Docker (DinD)，需要正确配置 ``services``。

### 2. 环境隔离：



- 在白名单中执行 `docker-compose` 时，确保正确配置 `.env` 文件或其他环境变量，以便将服务部署到正确的环境。

### 3. 错误处理：

- 在 `deploy_job` 中，可以添加更多脚本以处理部署失败的情况，例如日志收集或通知。

## 完整 `.gitlab-ci.yml` 示例（含 Docker-in-Docker）

```
stages:
  - build
  - deploy

variables:
  WHITELIST: "namespace-1,namespace-2,namespace-3"
  DOCKER_HOST: tcp://docker:2375/
  DOCKER_TLS_CERTDIR: ""

services:
  - docker:dind

build_job:
  stage: build
  script:
    - docker login -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD" $CI_REGISTRY
    - echo "Building Docker images..."
    - docker-compose build

deploy_job:
  stage: deploy
  script:
    - echo "Deploying using Docker Compose..."
    - docker-compose up -d
  rules:
    - if: '$CI_PROJECT_NAMESPACE && $WHITELIST =~ (?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)'
      when: on_success
    - when: never
```



以上配置结合 Docker Compose 和白名单控制，可以安全地实现命名空间限定的部署流水线。